Facultad de Ciencias

Despliegue de una plataforma big data de inteligencia de ciberseguridad basada en soluciones abiertas de compartición de información de amenazas

(A big data platform for cybersecurity intelligence based on open solutions of threat information sharing)

Trabajo de Fin de Máster
para acceder al

MÁSTER EN INGENIERÍA INFORMÁTICA

Autor: Álvaro Sainz Bárcena

Director\es: Enrique Vallejo y Roberto Hidalgo

Julio - 2020

Índice general

Índice de figuras

Agradecimientos

Quisiera dedicar este apartado a todas las personas que han hecho posible el presente trabajo y me han ayudado durante la realización del mismo.

Primero a mi familia por ofrecerme la oportunidad de poder continuar los estudios, y a mis amigos quienes han logrado que los eternos ratos de biblioteca sean hasta divertidos.

Además, a mi profesor y director Enrique por dedicar su tiempo en guiarme y aconsejarme en la redacción de este informe de una manera tan correcta y comprometida.

Por último, a mis compañeros de CIC con quienes he aprendido tanto, especialmente a José Manuel y José Luis, siempre disponibles para enseñarme y ayudarme en todo lo que he necesitado. También a Roberto por ofrecerme la oportunidad y confianza en la realización de este proyecto.

Resumen

La transformación digital llevada a cabo en prácticamente todas las empresas u organismos para conseguir sus propósitos supone un aumento de la superficie susceptible de ciberataques. La infraestructura y dispositivos tecnológicos que sostienen la tecnología de información, así como los propios datos, se han convertido en activos esenciales cuyo fallo o robo puede implicar daños irreparables. Para mejorar la ciberseguridad es de vital importancia disponer de conocimiento sobre un peligro existente que afecte a los activos para poder valorarlo y tomar las medidas oportunas, lo que se conoce como inteligencia de amenazas.

Este trabajo presenta el diseño y despliegue de una herramienta de ciberseguridad basada en tecnología Elasticsearch con la finalidad de mejorar la inteligencia de amenazas de una organización apoyándose en plataformas abiertas de información de amenazas. Esta herramienta permite enriquecer la información de amenazas proveniente de fuentes y sensores de ciberseguridad, tales como IPS (Intrusion Prevention System) o tráfico de red para dar una información de mayor calidad a los operadores y analistas de ciberseguridad.

Desde el punto de vista tecnológico, se ha construido una infraestructura en clúster con alta disponibilidad empleando virtualización en sistemas Linux donde se ha desplegado el motor Elasticsearch, un componente de ELK (Elasticsearch, Logstash y Kibana) que permite almacenar y buscar entre una gran cantidad de datos de manera muy eficiente. La herramienta permite una recolección masiva y cercana al tiempo real de diferentes fuentes de información, el filtrado, visualización y búsqueda ágil de información a través de un interfaz web disponiendo de medidas de autenticación.

Palabras clave: ciberseguridad, clúster, inteligencia de amenazas, virtualización, ELK.

Abstract

Digital transformation carried out in practically all companies or organizations to achieve their purposes involves an increase in the area susceptible to cyber attacks. The infrastructure and technological devices that support information technology, as well as the data itself, have become essential assets whose failure or theft may involve irreparable damage. In order to improve cybersecurity, it is vital to have knowledge about an existing danger that affects assets in order to assess it and take the appropriate measures, known as threat intelligence.

This work presents the design and deployment of a cybersecurity tool based on Elasticsearch technology with the aim of improving the threat intelligence of an organization by relying on open threat information platforms. This tool enriches the threat information from cybersecurity sources and sensors, such as IPS (Intrusion Prevention System) or network traffic, to provide higher quality information to cybersecurity operators and analysts.

From a technological point of view, a clustered infrastructure with high availability has been built using virtualization in Linux systems where the Elasticsearch engine has been deployed, an ELK component (Elasticsearch, Logstash and Kibana) that allows to store and search among a big data very efficiently. The tool allows a massive and close to real-time collection of different information sources, the filtering, visualization and agile search of information through a web interface with authentication measures.

Keywords: cybersecurity, cluster, threat intelligence, virtualization, ELK.

Capítulo 1

Introducción

El gran avance de las tecnologías de la información ha supuesto una importante evolución en los procedimientos y modelos de negocio de las empresas, empleando la transformación digital en ámbitos como el almacenamiento de los datos o la transferencia de información. El principal motivo de este cambio es debido a que las nuevas tecnologías proporcionan ventajas fundamentales a las compañías como permitir ofrecer mejores servicios, procesar los datos de manera mucho más eficiente o facilitar la conexión entre todos los sistemas independientemente de su ubicación. Las organizaciones emplean estos procesos ya que consiguen obtener un aumento significativo en la productividad y rentabilidad.

Como consecuencia, toda la infraestructura y dispositivos tecnológicos empleados para sostener la tecnología de la información, así como los propios datos, se han convertido en activos esenciales e imprescindibles. Todos estos componentes son susceptibles de ser atacados deliberada o accidentalmente con efectos perjudiciales para las empresas, independientemente de su tamaño. Los ciberataques pueden tener importantes consecuencias como la interrupción de producción, pérdida o robo de información confidencial, impacto negativo en la calidad de los productos, daño a la propiedad física o daño a la vida humana, pudiendo llegar a ocasionar daños irreparables. Por lo tanto, resulta necesario proteger estos activos a través del tratamiento de amenazas que ponen en riesgo la información que es procesada, almacenada y transportada por los sistemas de información que se encuentran interconectados. Este trabajo recibe el nombre de ciberseguridad [6].

Una de las tareas imprescindibles en ciberseguridad consiste en la monitorización tanto de posibles ataques o intrusiones, como del estado de funcionamiento de todos los activos que componen la infraestructura de tecnología de información ya que cualquier sistema o dispositivo conectado a una red puede presentar vulnerabilidades. Estos activos son sistemas heterogéneos que generan una gran cantidad de información de eventos y registros, dividiéndose principalmente en cuatro categorías: dispositivos de red (routers, switches, etc.), dispositivos de seguridad (firewalls, sistemas de detección y prevención de intrusos conocidos como IDS/IPS, etc.), servidores (web, mail, etc.) y aplicaciones. Por lo tanto, analizar todas estas distintas fuentes que originan tanta información es una tarea realmente complicada y tediosa para el equipo de ciberseguridad. Para solucionar este problema surgen los SIEM (Gestor de Eventos y de Seguridad de la Información), encargados de recolectar la información generada de fuentes de datos dispares, agregar y centralizar esta información en un almacén de datos, normalizarlos en un formato concreto para su visualización y producir alertas en tiempo real. Este tipo de herramientas son esenciales para ayudar a los analistas a visualizar los datos realmente relevantes entre la gran cantidad de información generada por los dispositivos de tecnología de información e identificar posibles peligros o amenazas.

Los ataques informáticos son cada vez más inteligentes, capaces y complicados de detectar debido al aumento de la complejidad de las nuevas tecnologías. Para lidiar con estos ataques innovadores es necesario disponer de un proceso de inteligencia de amenazas con el objetivo de adquirir conocimiento provechoso sobre los peligros para la organización en base a las fuentes de información disponibles. Sin embargo, conseguir esta información no es suficiente, sino que debe haber una comprensión de

cómo estos datos se relacionan con la organización, combinando múltiples fuentes de información con el entorno para determinar las amenazas relevantes. De este modo, la inteligencia de amenazas [7] se define como el proceso de adquirir conocimiento basado en evidencia, incluyendo el contexto, mecanismos, indicadores e implicaciones acerca de una amenaza existente o peligro para los activos que ayude al equipo de ciberseguridad a responder ante dicha amenaza o peligro. Integrar mecanismos de inteligencia de amenazas en un SIEM para conseguir un conocimiento de mayor valor mantendrá la infraestructura de tecnología de información mucho más segura.

1.1. Motivación del trabajo

El incremento de la utilización de nuevas tecnologías para llevar a cabo los propósitos de la empresa, vinculado al crecimiento de la complejidad de las mismas, supone un aumento de la superficie susceptible de ataques, siendo éstos cada vez más complicados de identificar, prever o evitar. Debido a las graves consecuencias que pueden ocasionar estos ataques, se requiere disponer de mecanismos actualizados que protejan a los activos de manera constante.

Para este objetivo, es de vital importancia adquirir y enriquecer el conocimiento sobre un peligro existente que afecte a los activos para poder valorarlo y tomar las medidas oportunas. Por lo tanto, es necesario disponer de una herramienta de ciberseguridad que permita mejorar la inteligencia de amenazas y poder hacer frente a ataques innovadores. De este modo surge el presente trabajo Fin de Máster, el despliegue de una plataforma de inteligencia de ciberseguridad basada en estándares abiertos de información de amenazas cuyo objetivo es identificarlas y aportar mayor conocimiento de las mismas. De esta manera, se proporcionará una información de mayor calidad a los operadores y analistas de ciberseguridad, apoyándoles a concretar las operaciones oportunas para hacer frente a los peligros existentes.

1.2. Objetivos del trabajo

La finalidad de este proyecto es desarrollar, configurar, integrar y desplegar una herramienta de ciberseguridad basada en tecnología ELK (Elasticsearch, Logstash y Kibana) que permita mejorar la inteligencia de amenazas de una organización apoyándose en plataformas abiertas de información de amenazas. Este sistema deberá garantizar una alta disponibilidad y ser tolerante a fallos. Para ello, se han propuesto una serie de objetivos:

- La plataforma deberá enriquecer la información de amenazas proveniente de fuentes y sensores de ciberseguridad, tales como IPS (Intrusion Prevention System), SIEMs (Security Information and Event Management) o tráfico de red.

- Desde el punto de vista tecnológico, se pretende construir una infraestructura en clúster empleando virtualización en sistemas Linux donde se despliegue el motor elasticsearch, un componente de ELK que permite almacenar y buscar entre una gran cantidad de datos de manera muy eficiente.

- La herramienta deberá permitir la recolección masiva y cercana al tiempo real de diferentes fuentes de información, el filtrado, visualización y búsqueda ágil de información a través de una interfaz web, disponiendo de medidas de autorización y autenticación.

De este modo, se pretende desarrollar dos campos. Por un lado, el despliegue de la infraestructura compuesta por los componentes software necesarios para garantizar los objetivos mencionados, que sostendrán los servicios de análisis de las amenazas. Y por otro lado, la integración y configuración de estos servicios en el sistema.

1.3. Estructura del documento

El contenido del presente informe está dividido en 6 capítulos incluyendo el actual. Los siguientes restantes se estructuran del siguiente modo:

- Capítulo 2: Diseño y herramientas empleadas. En este capítulo se describe el diseño general del sistema. También se detallan las tecnologías y herramientas utilizadas para componer la plataforma de inteligencia de ciberseguridad desarrollada.
- Capítulo 3: Despliegue de la infraestructura. Se explica el proceso seguido para construir la infraestructura en clúster con alta disponibilidad y la conexión entre todos los componentes que sostienen los servicios de ciberseguridad y el acceso a la herramienta.
- Capítulo 4: Integración con servicios de inteligencia de amenazas. En este capítulo se describen los mecanismos empleados para proporcionar inteligencia de amenazas a la herramienta.
- Capítulo 5: Resultados. Se exponen los resultados conseguidos y los paneles de visualización elaborados para la plataforma, así como la utilidad lograda de la plataforma.
- Capítulo 6: Conclusiones y trabajos futuros. En este último capítulo se explican las conclusiones obtenidas del trabajo realizado, así como posibles mejoras aplicables a la plataforma de inteligencia de ciberseguridad desarrollada.

Capítulo 2

Diseño y herramientas empleadas

En este segundo capítulo se explica el diseño de la plataforma desarrollada y las tecnologías utilizadas que componen la infraestructura informática para soportarla. Del mismo modo, se describen las principales herramientas software empleadas para elaborar el trabajo.

2.1. Diseño general del sistema

A continuación se expone una visión general de la solución desarrollada con las herramientas más relevantes, explicando el principal funcionamiento de la plataforma. El detalle y la motivación de este diseño se presenta más adelante, en los siguientes capítulos.

El sistema de ciberseguridad hace captura de tráfico de red, se analiza y se compara con fuentes externas de información permitiendo la generación de alertas. Para capturar el tráfico de red se pretende emplear un mini PC industrial, de modo que sea capaz de recolectar y analizar todo el tráfico que atraviesa un switch mediante agentes software. Posteriormente, este tráfico se compara con diversas fuentes externas de información de elementos peligrosos almacenadas en una base de datos local mediante Hippocampe, además de procesar su comportamiento con Suricata a través de un conjunto de reglas, para detectar e identificar posibles amenazas.

El tráfico de red analizado es enviado a un ELK (Elasticsearch, Logstash y Kibana), donde se parsea, almacena y visualiza toda la información. El acceso a estos datos se protege con medidas de autenticación a través de la herramienta Keycloak. En cuanto a la infraestructura empleada, la plataforma está desplegada en un entorno de virtualización formado por cuatro máquinas virtuales donde se consigue una alta disponibilidad del acceso a los datos tolerando la caída de uno de los dos nodos físicos que hospedan las máquinas virtuales.

A pesar de que el diseño inicial era emplear un mini servidor como dispositivo recolector del tráfico de red, se ha terminado empleando una RaspberryPi como alternativa debido a la falta de disponibilidad de éste.

2.2. Equipos y virtualización

La virtualización es una de las tecnologías más significativas que mayor impacto ha tenido en la computación durante los últimos años, con décadas de desarrollo y crecimiento. Su uso se expande cada vez más en las compañías debido a los valiosos beneficios que proporciona, destacando la consolidación de la infraestructura, disminución de costes, mayor seguridad, simplificación de la administración o el aumento de la productividad de los empleados, entre otros. A continuación se introducirá el concepto de esta tecnología sin explicar sus aspectos técnicos en detalle.

En computación, la virtualización se refiere a la creación de una versión virtual de algún recurso tecnológico, como una plataforma hardware, un sistema operativo, un dispositivo de almacenamiento

o recursos de red, a través de un software. Esto se consigue mediante el desacoplamiento del hardware manejado por un sistema operativo en una máquina física. En el caso actual, se empleará el tipo de virtualización de servidor o de plataforma hardware, con el propósito de simular un sistema operativo completo como si estuviera instalado en una plataforma hardware autónoma. Dicho de otra manera, esta virtualización se puede entender como un ordenador dentro de un ordenador implementado en software. Una instancia de un sistema operativo ejecutándose en un entorno virtual se denomina máquina virtual. Esta tecnología permite que múltiples máquinas virtuales con sistemas operativos heterogéneos se ejecuten simultáneamente y de manera aislada sobre una misma máquina física (denominada *host* o anfitrión). Virtualizando un sistema hardware completo, desde el procesador hasta la tarjeta de red, cada máquina virtual puede compartir un conjunto común de hardware sin llegar a ser consciente que otra máquina virtual también puede utilizar ese hardware al mismo tiempo. Sin embargo, el sistema operativo ejecutado en la máquina virtual ve un conjunto de hardware consistente y normalizado independientemente de los componentes hardware físicos.

En cuanto a la terminología relacionada con la virtualización, es importante conocer los conceptos esenciales básicos necesarios para componer esta tecnología. Como se ha mencionado anteriormente, el dispositivo conocido como *host* es el servidor físico que ejecuta el software de virtualización, contiene y comparte los recursos físicos como CPU, memoria, espacio de discos duros o tarjetas de red, entre otros, utilizados por las máquinas virtuales. Cada máquina virtual se denomina *guest*, siendo la representación virtualizada de una máquina física, ejecutada y mantenida por el software de virtualización que se comporta como si se ejecutara en un ordenador físico individual no virtualizado.

Un componente esencial de la virtualización es el denominado hipervisor o monitor de máquina virtual. Consiste en una capa de abstracción que implementa el software de virtualización, y se encuentra entre el hardware del host y las máquinas virtuales formadas por hardware y software virtualziado. El hipervisor se encarga de gestionar, manejar y arbitrar los recursos de la máquina física, repartiéndolos dinámicamente entre todas las MVs (Máquinas Virtuales) ejecutándose en el host. Esto permite que múltiples sistemas operativos (guests) se coordinen en el acceso simultáneo a los recursos hardware de una máquina virtual de un modo eficaz y sin conflictos.

Esencialmente existen dos tipos de hipervisores: el tipo 1, conocido como nativo, *unhosted* o bare-metal, y el tipo 2, también denominado *hosted*. Los hipervisores de tipo 1 se ejecutan directamente sobre el hardware físico, sin necesidad de la intervención de un sistema operativo que le aporte el acceso a los recursos hardware. Este software hace las funciones tanto de sistema operativo (con características mínimas) como de software de virtualización. Se emplea principalmente en organizaciones que disponen de servidores dedicados en exclusiva a la virtualización, ya que están optimizados para la ejecución de máquinas virtuales, emplean poca memoria RAM y consiguen poco overhead. Algunos ejemplos de este tipo de hipervisores son VMware ESXi, Xen o Microsoft Hyper-V, orientados a entornos de producción. Para el desarrollo del actual trabajo se ha empleado este tipo de hipervisor, cuya arquitectura se muestra en la figura 2.1.

Por otro lado, los hipervisores tipo 2 se ejecutan sobre un sistema operativo completo, cargado antes del hipervisor. Esta forma de virtualización es menos eficiente debido al overhead del sistema operativo host, y se emplea en entornos de desarrolladores donde se requiera probar software o para un uso doméstico, donde la seguridad y el rendimiento no es tan importante. Algunos de los más populares son Oracle VirtualBox o VMware Workstation.

2.2.1. Beneficios de la virtualización

La tecnología de virtualización ofrece numerosas ventajas que justifican su gran crecimiento en las empresas y organizaciones. Proporciona una solución viable para aumentar la productividad y reducir los costes de la infraestructura de TI, permitiendo controlar las cargas de trabajo en los centros de datos y disminuir el consumo de energía. A continuación se describirán algunos de los beneficios más esenciales que ofrece esta tecnología a organizaciones de todos los tamaños:

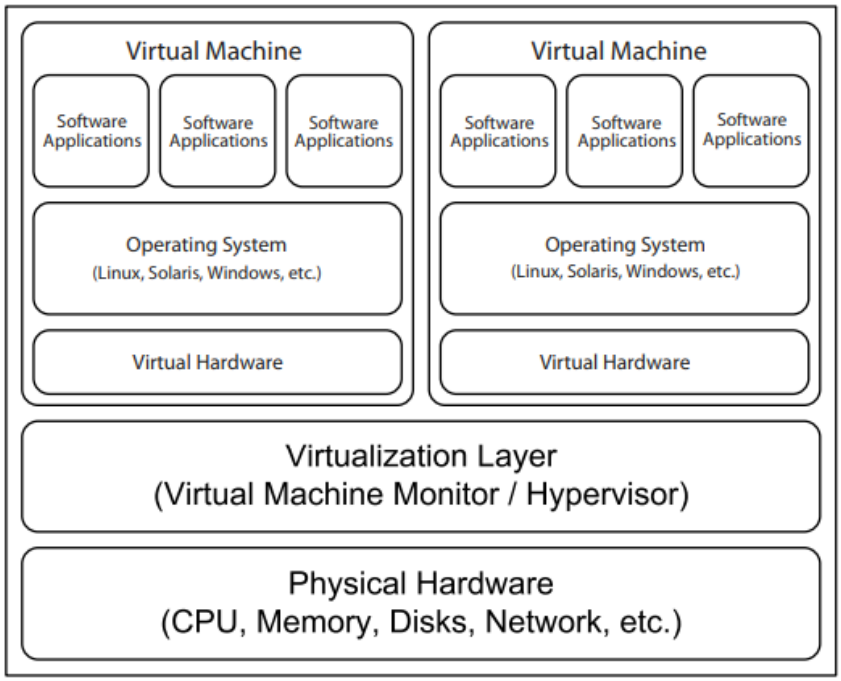


Figura 2.1: Descripción de la arquitectura del hipervisor tipo 1. Fuente: [1].

- Rápido tiempo de recuperación. En caso de fallo o caída de un sistema, la virtualización permite una recuperación más rápida de los recursos de TI, lo que proporciona una mejor continuidad del negocio e ingresos, disminuyendo el tiempo de inactividad del sistema informático.

- Escalabilidad. Los entornos virtualizados están diseñados para ser escalables, ofreciendo una mayor flexibilidad cuando se necesita aumentar los recursos hardware. Además, es posible aumentar estos recursos sin la necesidad de desconectar los servidores.

- Mayor seguridad. El entorno de cada máquina virtual se encuentra completamente aislado de la máquina host y del resto de máquinas virtuales, lo que permite construir entornos seguros personalizados a medida de las especificaciones de las aplicaciones ejecutadas.

- Portabilidad. Las máquinas virtuales se encapsulan en ficheros, lo que permite mover sistemas completamente configurados de un servidor físico a otro sin pérdida de datos.

- Sistema de testing. La virtualización permite desplegar entornos similares a escenarios de producción para desarrollar y probar aplicaciones software. Además, posibilita medir el rendimiento de éstas y determinar los recursos hardware óptimos para cada escenario específico, así como estudiar el comportamiento ante situaciones de fallo.

- Ahorro en costes y espacio. La virtualización permite disminuir los costes de la infraestructura de TI y su mantenimiento, ya que se necesita disponer de menos hardware debido a un uso más eficiente de estos recursos. Además, se consigue reducir el consumo de energía, así como facilitar la gestión/administración de los sistemas, lo que implica un menor número de personal de TI. Esto ocasiona un mejor retorno de la inversión (ROI).

Por otro lado, hay que tener en cuenta que la virtualización puede no funcionar bien en algunos entornos y presenta algunas limitaciones. Emplear esta tecnología supone un cierto *overhead*, por lo que el rendimiento disminuye ligeramente y no es apropiada para aplicaciones de alto rendimiento con uso intensivo de recursos. Tampoco es adecuada para aplicaciones que requieran un hardware específico, como dispositivos hardware personalizados debido a posibles incompatibilidades.

2.2.2. Servidores físicos e Hipervisores empleados

A continuación, se detallan los servidores físicos o nodos empleados donde se ejecuta el sistema desplegado, así como los hipervisores de cada uno de ellos mediante los que se han creado y administrado todas las máquinas virtuales. Se utilizan dos hipervisores distintos por motivos de seguridad, evitando que una posible vulnerabilidad del mismo afecte a los dos servidores.

Se han utilizado dos servidores físicos *Dell Inc. PowerEdge R630*; uno de ellos con el hipervisor Hyper-V de Microsoft y el otro con ESXi de VMWare para desplegar un total de cuatro máquinas virtuales.

Por un lado, las características principales del servidor Hyper-V son las siguientes:

- Procesador: 2 sockets Intel(R) Xeon(R) CPU E5-2680 V3 @ 2.50GHz, con 12 núcleos por socket, formando un total de 24 cores físicos (48 lógicos con hyperthreading).
- Memoria RAM: 192GB.
- Disco local: 4TB.
- Red: Intel Ethernet Server Adapter I350-t 1GbE
- Sistema Operativo: Microsoft Windows Server 2012 R2 Datacenter

Por otro lado, el servidor con VMWare ESXi dispone de:

- Procesador: 2 sockets Intel(R) Xeon(R) CPU E5-2690 V3 @ 2.60GHz, con 12 núcleos por socket, formando un total de 24 cores físicos (48 lógicos con hyperthreading).
- Memoria RAM: 128GB.
- Disco local: 1,64TB.
- Red: Broadcom NetXtreme BCM5720 1GbE
- Sistema Operativo: Sin sistema operativo externo. El hipervisor ESXi dispone de las principales funciones de un SO con un kernel Linux modificado para la ejecución del hipervisor y los componentes de virtualización de VMWare.

Tanto Hyper-V como VMWare ESi son hipervisores de tipo 1 o bare-metal, por lo que están orientados a ofrecer el mejor rendimiento en entornos virtualizados sin el overhead de un sistema operativo. Cuando se instala Hyper-V desde un sistema operativo Windows, la capa del hipervisor se coloca directamente sobre el hardware del host, y el Windows Server 2012 R2 Datacenter se convierte en una máquina virtual principal de manera transparente. Se han empleado dos servidores con el objetivo de disponer de un sistema tolerante a fallos, de tal manera que si uno de los servidores presenta algún fallo o se desconecta, el sistema informático continúe funcionando con total normalidad y sin pérdida de datos.

2.2.3. Máquinas Virtuales

Para el despliegue de la plataforma informática se van a emplear un total de cuatro máquinas virtuales, dos de ellas en el nodo de VMWare ESXi y las otras dos en el Hyper-V, de modo que la caída de las MVs de uno de ellos no implique ninguna parada en el funcionamiento del sistema. Por decisiones de diseño, cada MV actúa como un rol (datos o cliente) y con cuatro MVs se garantiza redundancia en cada uno de los roles. Además, la tecnología empleada para desplegar la plataforma consigue un mejor rendimiento con MVs de tamaño medio, y los servidores no disponen de tanto hardware libre para alojar más máquinas virtuales de este tipo. Dos máquinas harán funciones de cliente y las otras dos de almacenamiento de datos (se ve en detalle en el capítulo 3). El sistema

operativo (guest) empleado en todas las máquinas virtuales es un CentOS 7, ya que proporciona la compatibilidad necesaria con todas las herramientas software empleadas que componen la plataforma. Además, es un software libre y de código abierto, gratuito y fácil de instalar basado en la distribución empresarial Red Hat Enterprise Linux (RHEL). La distribución de los recursos asignados a cada máquina virtual se justifican con detalle en el capítulo 3, y son los siguientes:

Por un lado, las máquinas virtuales de datos (se denominarán máquinas 1 y 2) disponen de:

- CPU: 8 cores virtuales.
- Memoria RAM: 32GB.
- Almacenamiento: Dos discos duros virtuales, uno para el sistema operativo de 30GB y otro para los datos de 300GB. Al igual las MVs cliente, el primer disco se divide en tres particiones: una para `'/boot'` de 500MB, otra para el espacio de swap de 4GB, y la última para `'/'` de 25,5GB. En el disco de datos se crea una única partición en `'/datos'` de 300GB.

Por otro lado, las máquinas virtuales (se denominarán máquinas 3 y 4) cliente presentan:

- CPU: 4 cores virtuales.
- Memoria RAM: 8GB.
- Almacenamiento: Dos discos duros virtuales, uno para el sistema operativo de 50GB y otro para los datos de 100GB. A su vez, el primer disco se divide en tres particiones: una para `'/boot'` de 500MB, otra para el espacio de swap de 4GB, y la última para `'/'` de 45,5GB. En el disco de datos se crea una única partición en `'/datos'` de 100GB. Este particionado consigue mayor seguridad y aislamiento en los espacios, de modo que en caso de que una partición alcance el 100 % de ocupación, no afecte al resto de particiones.

Como se ha mencionado anteriormente, la virtualización permite disponer de múltiples máquinas virtuales y distribuir los recursos hardware en función de la carga y desempeño de las aplicaciones que soportan, consiguiendo un uso eficiente de éstos y con posibilidad de modificarlos si fuera necesario. El reparto de las cuatro máquinas virtuales permite conseguir una infraestructura con alta disponibilidad tolerante a fallos, ya que si se falla una MV cliente, una MV de datos, o un servidor físico, las otras MVs pueden continuar ejecutando el sistema informático sin pérdida de datos. Esta configuración se explica en el capítulo 3. Otra de las ventajas fundamentales de tener múltiples máquinas virtuales es la posibilidad de distribuir la carga entre todas ellas, evitando la saturación en cada una de ellas ante situaciones de alta carga y permitiendo un mayor número de conexiones simultáneas.

2.3. Herramientas software

En esta sección se explicarán brevemente los principales componentes software utilizados para componer la plataforma desarrollada. Todo el software empleado es libre y de código abierto, disponible en los repositorios de GitHub.

2.3.1. Elasticsearch

Elasticsearch es una de las herramientas más importantes que forman el sistema. Consiste en un motor de búsqueda basado en Lucene [8] y análisis distribuido que se encarga de almacenar los datos mediante el uso de objetos JSON. Esto y su API REST facilitan la integración con otras herramientas y lenguajes de programación. Permite la tecnología en clúster, es decir, se pueden desplegar varias instancias (nodos) de elasticsearch comportándose como un único nodo con mayor potencial. Los datos se almacenan en índices, estructuras que pueden estar compuestas por uno o más fragmentos (*shards*) de manera distribuida entre los nodos. Esto permite realizar búsquedas muy eficientes con un tiempo

de respuesta ínfimo sobre una gran cantidad de datos (paralelizando operaciones) y ofrecer una escalabilidad horizontal y dinámica. Además, cada fragmento puede tener una o más réplicas consiguiendo una alta disponibilidad y permitiendo un rebalanceo automático de los *shards* en caso de que falle algún nodo.

Otra de las principales razones del uso de elasticsearch es su fácil integración con logstash y kibana, formando el conjunto conocido como ELK Stack. Esta plataforma permite recolectar, filtrar/modificar y enviar información a elasticsearch a través de logstash, y visualizarla mediante kibana consultando a elasticsearch. De esta manera, se consiguen los elementos necesarios para construir un SIEM (Sistema de Gestión de Eventos e Información de Seguridad).

2.3.2. Logstash

Es una herramienta de administración de logs que forma parte de la pila ELK encargada de recopilar datos de múltiples fuentes de información simultáneamente, transformarla y enviarla a diferentes salidas. El procesamiento (pipeline) de los eventos de logstash se compone de tres fases: entrada → filtrado → salida.

En la primera fase, logstash recibe datos y genera eventos (información estructurada en campo:valor) de los datos recibidos por todas las fuentes al mismo tiempo, de forma continua. Tiene la capacidad de admitir fuentes de diversos sistemas en diferentes formatos, como por ejemplo eventos de una cola redis, eventos de kafka, http, tcp o udp, syslog, ganglia, lectura de un fichero del sistema, etc. Durante la fase de filtrado, se modifica (transforma) cada evento para conseguir la estructura y formato deseado. Esto se realiza a través de un conjunto de plugins que ofrecen una multitud de posibilidades (por ejemplo, añadir nuevos campos manualmente o aplicar expresiones regulares para realizar un parseo en un único mensaje para dividirlo en campos). Por último, se redirigen los eventos *parseados* a uno o varios outputs indicados. En este caso, se enviarán a la base de datos elasticsearch, pero se admiten muchas otras salidas, como un fichero en disco, un endpoint http(s), envío sobre TCP/UDP, o un servidor zabbix, entre otras.

2.3.3. Kibana

Kibana se utiliza para visualizar y explorar en tiempo real los datos almacenados en elasticsearch. Permite realizar filtros sobre los datos para obtener la información concreta deseada y la elaboración de paneles de visualización (*dashboards*). Cada *dashboard* se puede personalizar con distintos tipos de gráficos o representaciones de los datos. Esta herramienta es un cliente de elasticsearch, por lo que está limitado a realizar consultas únicamente a esta base de datos.

2.3.4. Cerebro

Esta herramienta web permite monitorizar y administrar clústers elasticsearch. Proporciona información general del estado de elasticsearch, como el número total de índices y shards o el almacenamiento toatal de los datos, y también del estado de cada nodo que compone el clúster: utilización de cpu, cantidad de memoria consumida, espacio en disco utilizado o su rol dentro del clúster. Además, es posible gestionar y visualizar los índices existentes de elasticsearch.

2.3.5. ElastAlert

ElastAlert es un framework encargado de generar diversas alertas sobre los datos almacenados en elasticsearch. Su funcionamiento consiste en realizar consultas periódicamente a la base de datos en tiempo real y comprobar si se cumple alguna de las reglas definidas. En caso afirmativo, se realizan las acciones determinadas para dicha regla. La principal ventaja de este plugin es que permite elaborar distintos tipos de reglas de manera flexible, como comprobar la frecuencia con la que se reciben los eventos, verificar si algún campo está incluido en una lista blanca/negra, conocer si el ratio de los eventos recibidos aumenta/disminuye, o comprobar si algún campo tiene dos valores disferentes en

un período de tiempo. ElastAlert admite la integración con varios servicios para realizar las acciones deseadas cuando se cumpla una regla, como crear una incidencia en JIRA, enviar un Email o enviar un mensaje por Telegram, entre otros.

2.3.6. Keycloak

Es una solución software de gestión de acceso e identidad dirigida a aplicaciones y servicios modernos, permitiendo su protección a través de medidas de autorización y autenticación.

Esta herramienta presenta numerosas ventajas que justifican su elección. Por un lado, ofrece *Single-Sign On/Out* (SSO) o autenticación única, de modo que una vez se haya accedido a una aplicación con las credenciales correctas, se guarda la sesión del usuario evitando tener que volver a identificarse en el resto de aplicaciones hasta su desconexión en cualquiera de ellas. Por otro lado, tiene la capacidad de actuar como *Identity Brokering*, permitiendo al usuario autenticarse a través de proveedores de identidad externos así como redes sociales mediante el uso de protocolos OAuth 2.0, OpenID Connect o SAML 2.0. Además, Keycloak puede federar bases de datos de usuarios externas, soportando la integración con LDAP y Directorio Activo. De este modo, se guardan los datos de usuario que ofrezca el proveedor externo por demanda en el almacenamiento local. Otra de sus ventajas es que admite la tecnología en clúster, permitiendo balancear la carga entre las instancias de Keycloak y proporcionando alta disponibilidad.

2.3.7. Galera MariaDB

MariaDB Galera es un clúster con topología multi-máster activo-activo con replicación síncrona de la base de datos MariaDB. Las principales ventajas de Galera son que consigue evitar la pérdida de ningún dato ni transacciones en caso de caída de nodos, garantiza la replicación de la información en todos los nodos al instante y detecta automáticamente la unión y desconexión de los nodos. Además, gracias a su topología todos los nodos del clúster tienen la capacidad de ejecutar transacciones permitiendo la lectura o escritura en cualquiera de los nodos en cualquier instante.

Se empleará esta tecnología como base de datos donde se almacene la información del servidor de Keycloak. Keycloak utiliza por defecto la base de datos relacional H2 para el almacenamiento persistente, pero se recomienda no utilizarse en un clúster (ejecuta cada operación de base de datos en todos los nodos del clúster) y reemplazarla por otra tecnología más madura ya que no es viable en situaciones con mucha concurrencia [9].

2.3.8. Nginx

Nginx es un servidor web con capacidad de actuar como proxy inverso, destacado por alcanzar un alto rendimiento. A diferencia de su principal competidor, el servidor web Apache, es muy ligero puesto que consigue reducir la cantidad de memoria empleada a través de crear únicamente los hilos justos y necesarios para atender peticiones en un período corto de tiempo. Además, consigue un tiempo de respuesta más rápido y atender un mayor número de peticiones por segundo (principalmente para contenido estático) [10]. Este aumento de rendimiento tiene asociado una disminución de flexibilidad, ya que al contrario que Apache, Nginx no permite definir directivas de configuración mediante ficheros especiales *.htaccess*.

Otra ventaja de Nginx es que puede actuar como balanceador de carga, consiguiendo repartir el tráfico entrante entre varios servidores web y evitar la saturación de los mismos. Además, ejerciendo como proxy inverso permite mejorar la seguridad porque consigue ocultar la existencia o características del servidor origen (inaccesible desde internet), reducir la carga de éste cacheando el contenido o actuar como único punto de autenticación. Nginx se empleará como proxy inverso hacia la herramienta de visualización de datos Kibana.

2.3.9. HAProxy

Este software es un balanceador de carga y servidor proxy para aplicaciones basadas en TCP o HTTP con capacidad de conseguir una alta disponibilidad en el sistema. HAProxy destaca por su baja latencia y el uso eficiente de recursos tanto de memoria como de cpu. Su principal función es redirigir las peticiones de los usuarios finales a los servidores web disponibles balanceando la carga mediante algoritmos como round robin, hashes de IPs origen o la elección del servidor con con el menor número de conexiones. Además, permite detectar la disponibilidad de los servidores web destino, de modo que no se redirigirán peticiones a los servidores caídos. HAProxy se empleará en un clúster Maestro-Esclavo para conseguir una alta disponibilidad y mantener el sistema operativo ante la caída de un servidor HAProxy. El estado de los HAProxy se comprobará continuamente mediante la herramienta Keepalived.

2.3.10. Keepalived

Keepalived es un *daemon* que permite comprobar el estado de servicios o servidores dinámicamente y de manera adaptativa y asignar direcciones IP virtuales (flotantes) en función de ello a través del protocolo VRRP (*Virtual Router Redundancy Protocol*). Su funcionamiento consiste en definir a uno de los nodos del clúster (compuesto por un total de dos nodos en este caso) como maestro, de modo que se le asigna la IP virtual, mientras que el otro nodo actúa como esclavo hasta el momento en que el maestro se desconecte por algún motivo. Entonces, la IP virtual se asignará a la interfaz del nodo esclavo que ahora actuará como maestro. Una de las ventajas de esta herramienta es que permite verificar el estado de servicios específicos, como HAProxy, de modo que permite conseguir una alta disponibilidad ante el fallo del servicio del nodo maestro.

2.3.11. Hippocampe

Hippocampe es una pieza software creada por *TheHive Project* basada en python que consiste en un agregador de feeds o fuentes de amenazas, compuestas por un conjunto de elementos potencialmente peligrosos, como puede ser una dirección IP, una URL o un dominio. Es capaz de descargar el contenido de estas fuentes de internet regularmente y almacenarlas en una base de datos local de elasticsearch, de modo que permite realizar consultas a través de una API REST y buscar si un elemento concreto se encuentra en alguna de las fuentes. En caso afirmativo, Hippocampe emplea el llamado *hipposcore* para medir el grado de confianza del elemento consultado, que depende del nivel de confianza introducido manualmente en las feeds, la cantidad de fuentes que tienen el elemento, y la cantidad de tiempo que lleva el elemento guardado en la fuente.

2.3.12. Packetbeat

Esta herramienta es un analizador ligero de paquetes de red en tiempo real. Es un componente de ELK Stack, por lo que facilita la integración y el envío de datos tanto a Elasticsearch como Logstash. Su funcionamiento consiste en capturar el tráfico de red, decodificar los protocolos en nivel de aplicación, correlacionar las peticiones con sus respuestas en transacciones, extraer y añadir campos, y por último enviar el evento agrupado en formato JSON a Logstash o Elasticsearch. Packetbeat soporta e identifica en tiempo real diversos protocolos como ICMP, DHCP, DNS, HTTP, Mysql o PostgreSQL, entre otros.

2.3.13. Suricata

Suricata es un sistema de detección de intrusos (IDS) y un sistema de prevención de intrusos (IPS) en tiempo real, capaz de monitorizar la seguridad de la red. Para conseguirlo, inspecciona el tráfico y decodifica los paquetes detectando automáticamente los protocolos empleados. Suricata se basa en utilizar un conjunto de reglas y compararlas con el tráfico analizado de la red y tomar las acciones oportunas cuando se cumpla alguna de ellas, bien sea generando un tipo de alerta o bloqueando el tráfico. Se pueden crear reglas manualmente o integrarlas de proveedores disponibles en internet como *EmergingThreats*. Este software alcanza un alto rendimiento gracias a su soporte multithreading y

capturar el tráfico mediante la tecnología AF_PACKET, un método de adquisición de paquetes más eficiente que otros como PCAP. Además, emplea formatos como YAML y JSON para los input y output, lo que facilita la integración con ELK.

Capítulo 3

Despliegue de la infraestructura

En este capítulo se explica el funcionamiento del software empleado para la construcción de la infraestructura de la plataforma, así como la justificación de la configuración utilizada. Además, se señalan los principales problemas encontrados durante la instalación y conexión entre los componentes, así como su resolución.

La plataforma debe permitir la recolección de una gran cantidad de datos provenientes de diferentes fuentes, ser capaz de enriquecer esta información recibida, y poder visualizarla y analizarla de manera eficiente. Además, el sistema debe ofrecer la disponibilidad de los datos y el acceso a la herramienta y ante un fallo hardware. La solución OpenSource distribuida ELK cumple estos requisitos, y se presenta en la sección 3.1. También se debe proteger el acceso a la interfaz web, por lo que se ha empleado la tecnología en Clúster Keycloak para ofrecer mecanismos de autenticación (sección 3.2). Para conseguir una alta disponibilidad se ha utilizado HAProxy junto con Keepalived, explicado en el apartado 3.3, de modo que se redirijan las peticiones web a un nodo disponible. Todo este software se aloja repartido en cuatro máquinas virtuales, dos actuando con el rol de cliente o frontend y dos con rol de datos o backend. Cada servidor dispone de una MV cliente y otra de datos, de manera que ambos roles estén redundados y se tolere el fallo de uno de los nodos. Por último, para conseguir integrar una de las fuentes de datos (tráfico de red) se ha empleado una RaspberryPi 3b (en lugar de un mini servidor por limitaciones de disponibilidad) como agente recolector, ya que es necesario disponer de un dispositivo hardware conectado directamente al switch cuyo tráfico es analizado. Esta integración se detalla en la sección 3.4. La arquitectura diseñada se puede observar en las figuras 3.1 y 3.6.

3.1. Clúster ELK

Se ha empleado el conjunto ELK debido a que CIC trabaja con esta tecnología desde hace tiempo y tiene experiencia y profundos conocimientos que han conseguido agilizar el proceso de aprendizaje. Además, a diferencia de otros sistemas de administración y análisis de logs como Splunk o Loggly, ELK presenta una potente versión OpenSource, permitiendo su uso gratuito. Otra alternativa similar es Graylog, pero éste no consigue la flexibilidad para modificar o parsear la información que ofrece el componente Logstash de ELK.

Según se introduce en la sección 2.3.1, un clúster es un grupo de una o más instancias de nodos conectados entre sí, comportándose como un único nodo de mayor potencial. Un clúster de elasticsearch consigue este objetivo mediante la distribución de tareas, búsqueda e indexación de los datos entre todos los nodos del clúster.

A continuación se detallará la arquitectura empleada para crear el Clúster ELK (en la versión 6.6.2), formado por un clúster de elasticsearch y varias instancias de Logstash y Kibana. Todo este software se aloja repartido en las máquinas virtuales descritas en la sección 2.2.3, como se muestra en la figura 3.1.

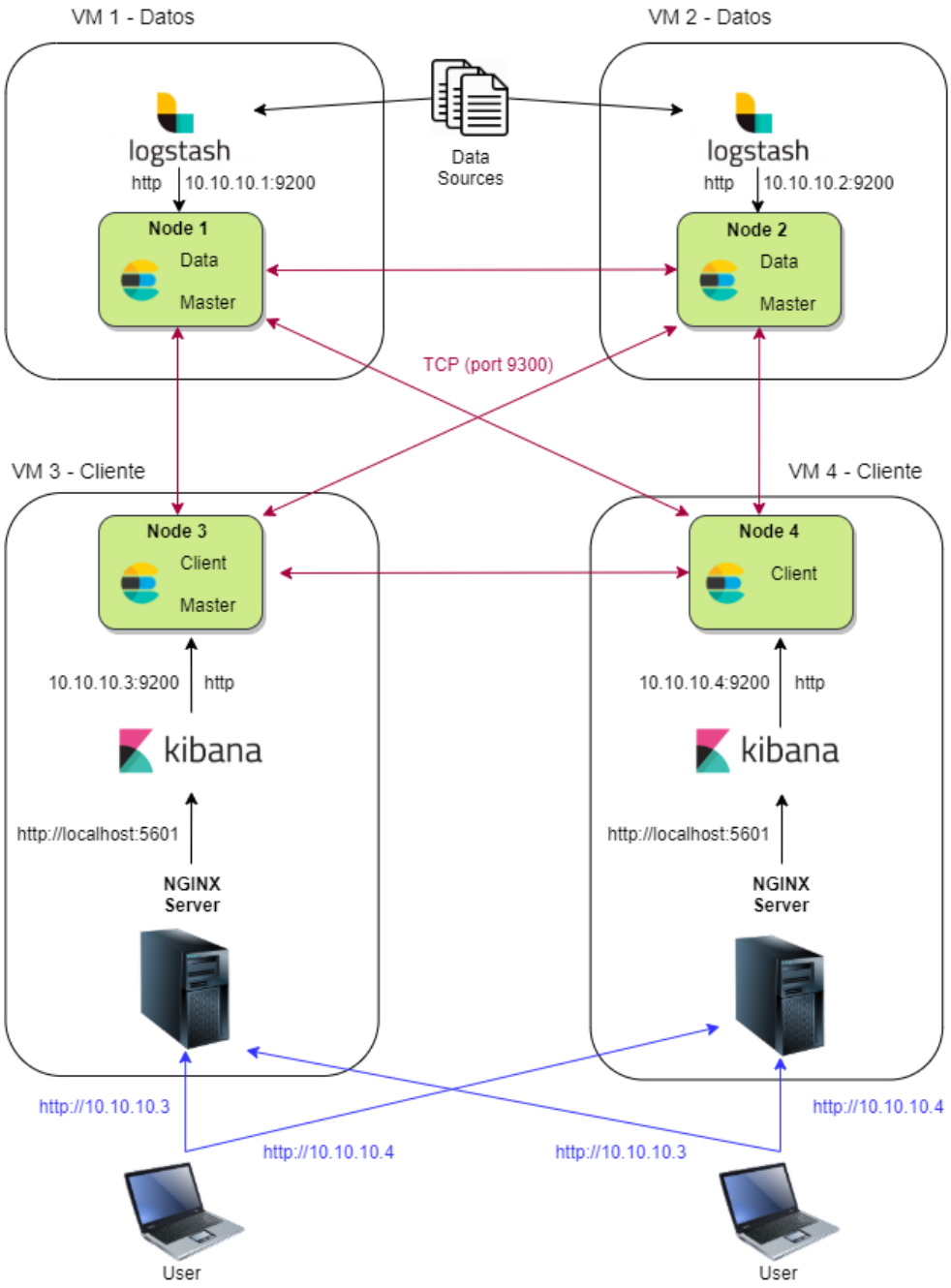


Figura 3.1: Arquitectura del sistema con Nginx.

3.1.1. Clúster Elasticsearch

Para desplegar y configurar el clúster se han seguido indicaciones de guías [3] [11] y la documentación oficial de Elasticsearch [12].

3.1.1.1. Tipos de nodos

Primeramente, es necesario conocer que a cada nodo o instancia de elasticsearch en el clúster se les puede asignar distintos tipos de labores o responsabilidades. Todos los nodos soportan tráfico HTTP y TCP de forma predeterminada, siendo esta última empleada exclusivamente para la comunicación entre los nodos del clúster. La capa HTTP es utilizada únicamente para las peticiones de clientes externos mediante el estándar de intercambio de datos REST. Cada nodo puede llevar a cabo uno o más de los siguientes propósitos:

- Data node: Nodo encargado de almacenar los datos (shards formados por los documentos que han sido indexados) y ejecutar operaciones *CRUD* (*Create*, *Read*, *Update*, *Delete*), de búsqueda y de agregaciones. Estas acciones son de uso intensivo en CPU, E/S (entrada/salida) y memoria, por lo que es conveniente monitorizar estos recursos y añadir más nodos de datos en caso de sobrecarga. La principal ventaja de emplear nodos de datos dedicados es la separación de las funciones de maestro (explicado a continuación) y de datos.

- (Eligible) Master node: Nodo susceptible de ser elegido como máster. Los nodos máster se encargan del control y gestión de todo el clúster, y realizan acciones de configuración como añadir o eliminar nodos del clúster, crear o eliminar índices, o decidir qué fragmentos (*shards*) se asignan a cada nodo.

- Ingest node: Elasticsearch permite definir una serie de procesadores o procesos encargados de transformar y enriquecer el documento antes de su indexación. Los nodos de ingesta son los encargados de realizar esta función. Dependiendo del tipo de operaciones realizadas por estos procesadores y los recursos requeridos, puede tener sentido emplear nodos de ingesta dedicados, llevando a cabo esta única tarea específica.

- Coordinating node: Distintas peticiones como una solicitud de búsqueda o de indexación masiva (múltiples operaciones de indexación, es decir, añadir un documento a un índice para permitir su búsqueda, o eliminación en una única llamada API) pueden involucrar datos almacenados en distintos nodos de datos. Una solicitud de búsqueda, por ejemplo, se ejecuta en dos fases que son coordinadas por el nodo que recibe la solicitud del cliente: el nodo coordinador.

  En la fase de dispersión (*scatter*) el nodo de coordinación reenvía la petición a los nodos de datos que contienen los datos. Cada nodo de datos ejecuta la solicitud localmente y devuelve sus resultados al nodo de coordinación. En la fase de recopilación (*gather*), el nodo de coordinación reduce los resultados de cada nodo de datos en un único conjunto de resultados global.

  Cada nodo es implícitamente un nodo de coordinación (y no se puede desactivar esta función), de modo que si un nodo tiene desactivadas las funciones de master, data e ingest, únicamente actuará como nodo de coordinación. Este nodo necesitará disponer de suficiente memoria y CPU para lidiar con la fase de recopilación. En definitiva, este rol puede definirse como un balanceador de carga inteligente, encargado de enrutar solicitudes, distribuir la indexación masiva de datos y tramitar la reducción de búsqueda.

Es muy importante para la estabilidad del clúster que los nodos máster realicen el menor trabajo posible. Indexar y buscar datos es un trabajo intenso en CPU, memoria y entrada/salida que puede ejercer presión sobre los recursos de un nodo, por lo que es conveniente en clústers grandes dividir los roles de máster y de datos en nodos dedicados a cada función. A pesar de no almacenar información, los nodos máster dedicados deben tener acceso al directorio de datos (al igual que los data nodes) porque es donde persiste el estado del clúster entre los reinicios de los nodos. Cualquier nodo maestro elegible puede ser seleccionado para convertirse en el nodo máster, determinado por el proceso de elección del máster [13].

Las funciones o roles asignados a cada nodo del clúser se definen en el fichero de configuración principal de elasticsearch de cada nodo. Por defecto, una instancia de elasticsearch realiza las funciones de (elegible) master, ingest y data. Esta configuración es apropiada para clústers de tamaño reducido, pero a medida que aumenta el número de nodos, es importante considerar la separación de los nodos master, de datos, y de ingesta (siempre y cuando éstos deban computar procesadores de alta carga). También es relevante definir nodos con estas tres labores desactivadas, de modo que existan nodos únicamente con la función de *coordinar*, los cuales serán empleados para recibir las peticiones externas HTTP, actuando como clientes y balanceadores de todo el clúster.

En este caso, como se explica en la sección 2.2.3, se disponen de cuatro máquinas virtuales, por lo que se ha creado un pequeño clúster de cuatro nodos de elasticsearch. En la imagen 3.2 se puede

observar la arquitectura del clúster empleada y las funciones asignadas a cada nodo. Se han establecido dos nodos de datos encargados de almacenar la información y otros dos clientes (coordinadores) que serán a quienes se les enviarán las peticiones HTTP. No se han establecido nodos de ingesta porque no se ha definido ningún preprocesador para enriquecer o procesar los datos. Esta tarea la realizará Logstash (se ve en la sección 3.1.2) ya que cuenta con una selección de complementos mayor para procesar los datos y su configuración es más sencilla. En cuanto al rol de elegible-master, se han establecido tres nodos para realizar esta función, permitir la caída de uno de ellos y también evitar el split brain (se explica a continuación en la sección 3.1.1.2).

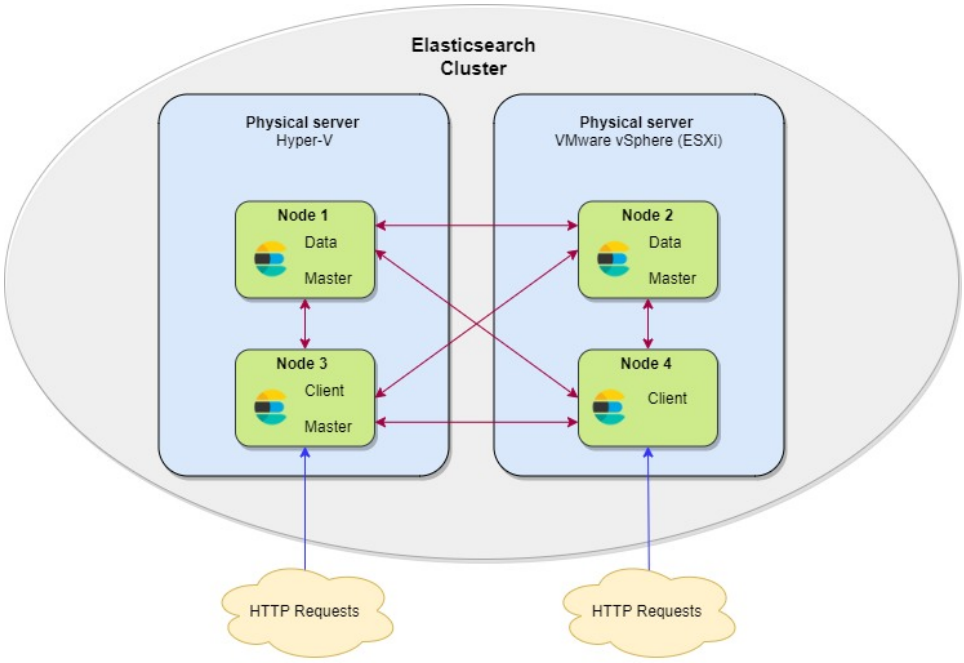


Figura 3.2: Arquitectura del clúster elasticsearch.

3.1.1.2. Evitar *Split Brain*

Cuando se emplea la tecnología en clúster, existe un problema crítico conocido como *split brain* que causa pérdidas de datos cuando se ocasiona en determinadas situaciones. Este contratiempo de un sistema distribuido ocurre cuando existen inconsistencias de datos originadas por un fallo de comunicación que ocasiona la separación de datos en dos conjuntos independientes. Para evitarlo, es de vital importancia configurar correctamente el parámetro *discovery.zen.minimum_master_nodes* mediante el fichero de configuración de elasticsearch (por defecto su valor es 1). Esto permite a los nodos elegible-máster conocer el número mínimo de nodos con el rol elegible-máster visibles necesarios para poder formar un clúster.

En el caso concreto de un clúster de Elasticsearch, este problema puede ocasionarse, por ejemplo, cuando está compuesto por dos nodos elegible-máster. Si un fallo en la red interrumpe la comunicación entre ambos nodos, cada uno ve un único nodo elegible para ser maestro (él mismo). Con el valor del parámetro *discovery.zen.minimum_master_nodes* predeterminado (1), cada nodo elegible-maestro se elige a sí mismo como maestro pensando que el otro nodo elegible-master ha caído. Como resultado, se forman dos grupos o un cerebro dividido, y no se volverán a juntar en un único clúster hasta que se reinicie uno de los nodos. Cuando esto ocurra, se perderán todos los datos que se hayan escrito desde el fallo de red en el nodo reiniciado.

En cambio, una posible situación en la que no ocurre este problema, sería en un clúster formado por tres nodos elegibles maestros y definido el parámetro del número mínimo de nodos elegible-master establecido a un valor de 2. En el caso de un fallo de red que separe un nodo de los otros dos, el lado compuesto por un nodo elegible-master no puede seleccionarse a sí mismo como máster ya que no

es capar de detectar el número mínimo de nodos elegible-master definido. Serán los otros dos nodos quienes negocien para que uno se convierta en maestro (si es necesario) y el clúster continuará funcionando correctamente. Cuando se solvente la división de la red, el nodo que había quedado aislado se integrará de nuevo en el clúster y comenzará a atender las solicitudes nuevamente.

El parámetro *discovery.zen.minimum_master_nodes* necesario para evitar el *split brain* o cerebro dividido depende del número de nodos asignados con el rol elegible-master en el clúster, y en Elasticsearch se calcula como: $\lfloor (master\_eligible\_nodes/2) \rfloor + 1$. En el caso anterior, quedaría: $\lfloor (3/2) \rfloor + 1 = 2$. Para permitir la caída de un nodo elegible-master mientras el clúster permanezca funcionando correctamente, se debe establecer al menos tres nodos elegible-master.

3.1.1.3. Índices y Shards

Como se menciona en la sección 2.3.1, Elasticsearch almacena los datos en forma de documentos JSON de manera distribuida, en lugar de emplear filas y columnas como las bases de datos relacionales. Un índice puede considerarse como una colección optimizada de documentos, y cada documento es un conjunto de campos formados por la pareja *llave-valor* que contienen los datos. Por defecto, Elasticsearch indexa todos los datos en cada campo y cada campo indexado tiene una estructura de datos optimizada y dedicada. Por ejemplo, los campos de texto se almacenan en índices invertidos, y los campos numéricos y geográficos se almacenan en árboles BKD [14]. Cuando se almacena un documento, éste se indexa, lo que permite ser buscado. Elasticsearch emplea una estructura denominada índice invertido que soporta búsquedas de texto completo muy rápidas. Un índice invertido registra cada palabra única que aparece en cualquier documento e identifica todos los documentos en los que aparece cada palabra. La rapidez de Elasticsearch reside en la capacidad de usar las estructuras de datos por campo dedicadas. Además, tiene la capacidad de no tener esquemas, de modo que los documentos pueden indexarse sin especificar explícitamente cómo manejar cada campo. Esta capacidad se denomina *mapping dinámico*, detectando y agregando automáticamente nuevos campos al índice asignándoles el tipo apropiado. Sin embargo, es posible definir *mappings estáticos*, indicando previamente a la indexación de documentos el tipo de los campos, para, por ejemplo, poder distinguir entre campos de cadena de texto completo y campos de cadena de valor exacto.

Es importante conocer ciertos aspectos de Lucene para escoger una correcta infraestructura donde desplegar Elasticsearch. Lucene es el motor de búsqueda en el que se basa esta tecnología. Cada índice de Elasticsearch se divide en *shards*, y cada *shard* es un índice de Lucene (permite un máximo de casi 2^{31} documentos cada uno). A su vez, cada índice de Lucene se divide en pequeños ficheros llamados segmentos (figura 3.3) los cuales son inmutables (nunca cambian). Cuando se añaden nuevos documentos a un índice de Elasticsearch, Lucene crea un nuevo segmento y lo escribe en disco.

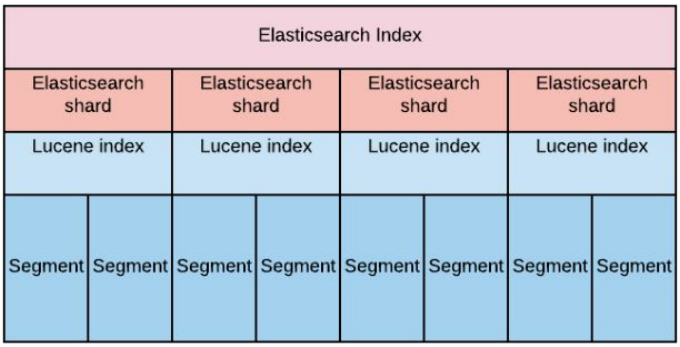


Figura 3.3: Segmentos Lucene. Tomada de [2].

Lucene realiza las búsquedas en todos los segmentos de manera secuencial, por lo que cuantos más segmentos se tengan, mayor será el tiempo de búsqueda. Para mitigar esta situación, periódicamente Lucene realiza operaciones *merge*, fusionando un conjunto de pequeños segmentos en uno mayor. Esta operación tiene un importante coste en CPU y E/S, por lo que se recomienda deshabilitar los *merges*

al realizar una indexación masiva. Si el nodo contendrá muchos *shards* y segmentos, sería apropiado escoger un sistema de ficheros con un buen rendimiento a la hora de manejar muchos archivos pequeños y no tenga una importante limitación de inodos. Otro aspecto relevante a tener en cuenta es que Lucene realiza *copy on write* cuando se actualiza o borra un documento, implicando que éste nunca se elimine, sino que se marca como eliminado y se crea otro en caso de actualizarse. Como consecuencia, los índices crecerán en disco hasta eliminarlos por completo. Para borrar los documentos marcados como eliminados se debe forzar un *merge* de los segmentos de Lucene. Durante un *merge*, Lucene toma dos segmentos y mueve el contenido a uno nuevo, borrando estos dos primeros del disco, por lo que se requiere al menos un espacio suficiente en disco equivalente al tamaño de los segmentos a fusionar. Esto puede causar un problema cuando se hace el *merge* de un *shard* enorme, ya que si su tamaño es mayor que la mitad del espacio disponible en disco, probablemente no se pueda realizar la operación por completo.

Como se mencionó anteriormente, los fragmentos o *shards* que componen cada índice se distribuyen entre los nodos del clúster. El número de *shards* que componen un índice se define a la hora de la creación del mismo. Es un parámetro importante que afecta al rendimiento, y existen *trades offs* con respecto al número de fragmentos primarios y su tamaño. Escoger el número de *shards* apropiado puede ser un problema porque normalmente es muy complicado conocer cuántos documentos se almacenarán en la base de datos antes de comenzar a recibirlos. Cuanto mayor sea el número de shards, existe más *overhead* en simplemente mantener el índice, pero cuanto más grande sea cada fragmento, mayor será el tiempo empleado para moverlos cuando se necesite rebalancear el clúster (Elasticsearch lo realiza de manera automática). Por otro lado, lanzar consultas sobre múltiples pequeños *shards* hace que el procesamiento sea más rápido, pero a su vez muchas consultas implican un *overhead*. Para establecer un óptimo número de *shards* por índice, habría que crear una prueba representativa de los datos y lanzar consultas realistas, determinando cómo afecta en rendimiento a medida que aumenta el tamaño del *shard*, y así averiguando en cuántos se debe componer el índice. Este proceso suele ser difícil de llevar a cabo; por norma general, y como el propósito de este sistema se basa en datos temporales, es apropiado un tamaño medio de entre 20GB y 40GB por fragmento [15]. Para este caso, se ha planeado crear un índice diario y no se espera que su tamaño sea mayor de 40GB, por lo que se creará un fragmento principal por cada índice.

Existen dos tipos de *shards*, los primarios y las réplicas, siendo estas últimas copias exactas de los primarios ofreciendo redundancia para no perder datos y conseguir alta disponibilidad en caso de fallo en algún data node. En ese caso, la réplica se convertiría automáticamente en primaria. El número de réplicas también se define a la hora de crear el índice, pero puede ser modificado posteriormente. Como en el presente clúster se disponen únicamente de dos nodos de datos, el número de réplicas por *shard* primario se ha establecido en 1. En la figura 3.4 se muestra un ejemplo de un clúster compuesto por dos nodos de datos, donde existe un índice formado por 5 *shards* y definida una réplica por cada fragmento.

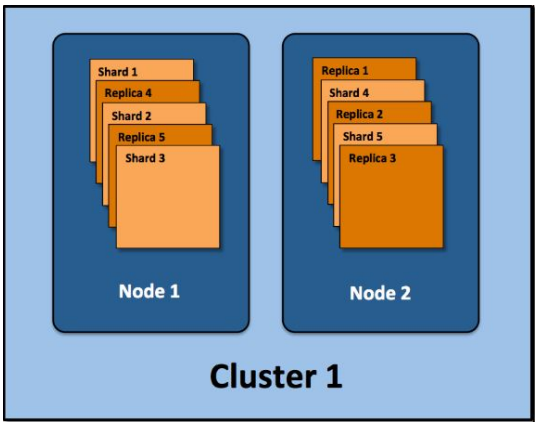


Figura 3.4: Distribución *shards* y réplicas. Imagen de [3].

3.1.1.4. Hardware

A la hora de montar un clúster de Elasticsearch es de vital importancia escoger el hardware apropiado para conseguir el mejor rendimiento. Como consideración general, lo mejor es utilizar máquinas de tamaño mediano, evitando así tener que administrar muchas máquinas pequeñas y, por otro lado, evitar un desequilibrio en el uso de los recursos del servidor (por ejemplo, consumiendo toda la memoria pero tener la CPU sin utilización). Además, la mayoría de los requerimientos hardware serán dependientes del throughput (número de eventos por segundo recolectados), la política de retención de datos y el tamaño medio de los eventos. En este subapartado se comentan los principales aspectos de los recursos hardware que se han tenido en cuenta: CPU, memoria, red y almacenamiento.

CPU

La ejecución de consultas filtradas complejas, operaciones CRUD, indexación intensiva o consultas contra un conjunto de caracteres no latinos tienen un fuerte impacto de CPU, así que elegir el procesador correcto es crítico. Como Elasticsearch está basado en Java, se debe estudiar las especificaciones del procesador para averiguar cómo se comporta con esta tecnología. Además, existen resultados del rendimiento de servidores y procesadores en aplicaciones basadas en java publicadas en Standard Performance Evaluation Corporation [16]. Gracias a que Elasticsearch escala bien horizontalmente (agregando nuevas máquinas), a pesar de que el escalado vertical (hardware más potente) puede proporcionar algunos beneficios, por lo general, el coste de las CPU masivas no proporciona tanta ganancia para merecer la pena. Elasticsearch obtiene un buen rendimiento con 4, 6 u 8 cores [11], por lo que no es necesario adquirir un procesador con más núcleos. En el caso actual, se emplean 4 cores para los nodos clientes y 8 para los de datos, ya que estos últimos tendrán mayor utilización de este recurso.

Memoria

La memoria de cada máquina virtual se divide en dos secciones: la usada por el sistema operativo (Linux en este caso) como buffer/cache de disco, y el espacio dedicado a Java usada por la aplicación (Elasticsearch), conocido como JVM (*Java Virtual Machine*). A su vez, la JVM está compuesta por el espacio heap, no-heap y otro espacio dedicado ciertos datos. La memoria heap es muy importante ya que es utilizada por muchas estructuras de datos en memoria para proporcionar una rápida operación en Elasticsearch. No obstante, Elasticsearch no solo se basa en la memoria heap, también hace uso del espacio no-heap para Lucene y todos los hilos (cada uno reserva 256KB) creados dentro de los thread pools [17] destinados a realizar distintas operaciones.

Lucene está diseñado para aprovechar el sistema operativo subyacente para almacenar en caché las estructuras de datos en memoria. Los segmentos de Lucene se almacenan en archivos individuales. Como estos segmentos son inmutables, los archivos nunca cambian, por lo que se ven muy favorecidos para aprovechar la caché y el SO los mantendrá calientes en memoria para un acceso más rápido. De esta manera, si se asigna demasiada memoria al espacio heap empleado por elasticsearch, no quedará mucho para Lucene y afectará seriamente al rendimiento. La recomendación estándar es asignar el 50 % de la memoria disponible a la heap, dejando la otra mitad libre, la cual irá consumiendo Lucene. De esta manera, al no disponer de una jvm muy grande, se obtendrá una mejora de rendimiento considerable ya que se reduce el impacto negativo causado por el recolector de basura de Java y además Lucene podrá disponer de más memoria para emplear como caché.

Hay otra importante razón para no asignar mucha cantidad de memoria heap y por la que se recomienda emplear menos de 32GB a ésta. En Java, todos los objetos se asignan en el espacio heap y se hace referencia a ellos mediante un puntero llamado oop (*ordinary object pointer*) de 32 o 64 bits dependiendo si se ejecuta en un sistema ILP32 o LP64, que hace referencia a la ubicación exacta del byte del valor. Esto implica que los sistemas de 32 bits (ILP32) solo pueden emplear 4GB de memoria dinámica. Para los sistemas de 64 bits este problema no ocurre, pero el *overhead* de estos punteros resulta en desperdiciar espacio simplemente por el tamaño del mismo. Además, los punteros más grandes consumen un mayor ancho de banda al mover valores entre memoria principal y las distintas cachés. Para mitigar este impacto, Java utiliza una técnica llamada *oops comprimido* [18], de

modo que en lugar de apuntar a localizaciones exactas de bytes en memoria, los punteros hacen referencia a los offsets de los objetos. Esto implica que un puntero de 32 bits pueda apuntar a cuatro mil millones de objetos, en lugar de cuatro mil millones de bytes, lo que significa que el almacenamiento dinámico puede crecer hasta alrededor de 32GB usando punteros de 32 bits. Una vez cruzado este límite, los punteros vuelven a ser oop, volviendo a ser de 64 bits y con el mismo problema. Debido a lo mencionado, a pesar de disponer de una máquina con memoria de sobra (más de 64GB), solo se debería asignar menos de 32GB (se recomienda 31GB) y dejar el resto para el espacio no heap, evitando perder memoria, disminuir el rendimiento de CPU, y aumentar el impacto del recolector de basura. Otra recomendación general es no emplear máquinas con menos de 8GB de RAM ya que se terminaría necesitando muchas máquinas pequeñas.

Por otro lado, cuanto mayor ancho de banda y menor latencia tenga la memoria, se conseguirá cierta mejora de rendimiento, pero Elasticsearch no es muy sensible en este aspecto por lo que no es de mucho interés invertir excesivamente en ello. En este caso, en las máquinas virtuales con rol de cliente se ha optado por asignar 8GB (4 para heap) ya que, al no manejar datos, no requieren tanta capacidad de memoria, mientras a los nodos de datos se les ha asignado 32GB (16 para heap).

Red

La red es un recurso muy importante para el rendimiento de un sistema distribuido, siendo necesario que sea rápida y confiable. Siempre que sea posible conseguir mayor ancho de banda y menor latencia en la red, será mejor para Elasticsearch. La baja latencia garantiza una buena comunicación entre los nodos, mientras que un ancho de banda alto ayuda en el movimiento o recolocación y recuperación de los *shards*. Por norma general, una red de 1GbE o 10GbE es suficiente para la gran mayoría de los clústers. No se recomienda emplear múltiples centros de datos para un mismo clúster, sobretodo si se encuentran a grandes distancias entre sí, ya que las latencias grandes suponen problemas en los sistemas distribuidos y dificultan la depuración. En este caso, como se menciona en la sección 2.2.2, las interfaces de red de ambos nodos disponen de un ancho de banda de 1 Gigabit Ethernet, siendo suficiente para alcanzar un buen rendimiento de red.

Almacenamiento

Los discos mecánicos son el recurso más lento de un servidor, por lo que en los nodos de backend (data nodes) pueden convertirse en el cuello de botella en clústers con alta capacidad de escritura. Los SSD consiguen un importante aumento de rendimiento tanto en las consultas como en la indexación, de modo que si se puede afrontar, se deberían emplear discos SSD. Esto es debido a que, además de la considerable menor latencia de los SSD en las operaciones de lectura/escritura respecto a los HDD, estos últimos obtienen un rendimiento muy bajo a la hora de manejar una gran cantidad de ficheros de tamaño reducido, como son los segmentos generados por elasticsearch a la hora de su indexación. Una mejor opción es emplear dispositivos SSD con interfaz NVM Express (más caros y con mejor rendimiento que los SSD SATA), ya que permite un mayor ancho de banda explotando el paralelismo respecto a los SSDs SATA.

Por otra parte, se recomienda [3] emplear medios locales en lugar de almacenamiento conectado en red (NAS) o una red de área de almacenamiento (SAN) debido a sus mayores latencias y puntos únicos de fallo, y emplear un volumen RAID 0 en caso de usar varios discos. Como Elasticsearch proporciona alta disponibilidad a través de las réplicas (en caso de disponer de un clúster), no hay necesidad de disponer de discos con información redundante en cada nodo. Por ello, RAID 0 proporciona una mayor capacidad de almacenamiento además de permitir el uso de todos los discos paralelamente consiguiendo un mejor rendimiento. En el caso actual, se dispone de un entorno virtualizado, se está empleando un único disco virtual para la partición de datos en cada máquina virtual, a partir de los medios locales del servidor físico formados en un RAID 5, y no se ha modificado porque existen otras máquinas virtuales en los servidores.

3.1.1.5. Instalación y Configuración del sistema

En este subapartado se explicarán los aspectos más relevantes tanto en el proceso de instalación y configuración de Elasticsearch como en la configuración del sistema operativo para la optimización de esta tecnología.

Como se dispone de un sistema operativo CentOS, la instalación de Elasticsearch se realiza mediante un fichero RPM disponible en la web [19], siguiendo los pasos indicados en la misma [20]. Una vez instalado manualmente en todas las máquinas virtuales, se deben aplicar ciertas configuraciones generales y preparar el cluster de modo que los nodos puedan conectarse y comunicarse entre sí. Para ello, se modifican algunos parámetros en el principal fichero de configuración de Elasticsearch `/etc/elasticsearch/elasticsearch.yml`. Estos son el nombre del clúster (común para los 4 nodos), el nombre del nodo en cuestión, el directorio donde se almacenarán los datos y los logs, y el tipo(s) de nodo. Además, en lo referente a la red, hay que especificar el parámetro `network.host` (por defecto es localhost) con una IP o hostname para publicar el host tanto para recibir peticiones como para anunciarse al resto de nodos del clúster. También se puede modificar el puerto asignado para recibir las peticiones http (por defecto es el 9200) y el puerto para la comunicación entre los nodos por tcp (por defecto el 9300). En este caso, se han mantenido los puertos por defecto y se ha publicado en la dirección IP cada máquina virtual para permitir la comunicación entre los nodos del clúster. Otro importante parámetro es el `discovery.zen.ping.unicast.hosts`, donde se deben definir las IPs del resto de nodos del clúster, y el `discovery.zen.minimum_master_nodes` comentado en el punto 3.1.1.2.

En lo referente a la memoria se deben aplicar ciertas configuraciones para conseguir un buen rendimiento. Una de ellas es definir el tamaño del espacio de memoria heap mínimo (parámetro Xms) y máximo (parámetro Xmx) especificándolo en el fichero `/etc/elasticsearch/jvm.options`, siendo recomendable especificar el mismo valor en ambos parámetros. En este caso, se han establecido 16GB (mediante los parámetros Xms16g y Xmx16g) para la heap de JVM en los nodos de datos de Elasticsearch (MVs 1 y 2) y 4GB (Xms4g y Xmx4g) en los clientes (MVs 3 y 4). Otra importante configuración a realizar es desactivar el *swapping*, un proceso llevado a cabo por la mayoría de sistemas operativos mediante en cual se intercambia memoria no utilizada (incluso heap) por la aplicación a un espacio reservado en disco (swap). Este proceso afecta negativamente al rendimiento y estabilidad del nodo de elasticsearch, pudiendo llegar a prolongar altamente la duración del algoritmo de recolección de basura. Para asegurar que la memoria del proceso de elasticsearch no se intercambia, se debe activar la función *mlockall*, de modo que todas las páginas asignadas por el espacio de direcciones del proceso residirán en memoria hasta que se desbloqueen o hasta que el proceso finalice. Un último aspecto importante a tener en cuenta son los descriptores de archivos, ya que Lucene usa una gran número de ficheros (se crean muchos ficheros para administrar la indexación paralela en el mismo *shard*) y Elasticsearch muchos sockets para la comunicación entre nodos o clientes HTTP, por lo que es necesario aumentarlos. Por defecto, la instalación mediante RPM ya establece un número suficiente de descriptores de fichero por proceso (65535), por lo que no se requiere configuración adicional.

Una vez aplicados los cambios para alcanzar un rendimiento satisfactorio, se deben establecer algunos cambios en la seguridad del sistema operativo para permitir la comunicación entre los nodos del clúster. Una de las principales funciones del firewall de CentOS 7 (servicio *firewalld*) es establecer un control sobre las conexiones entrantes y salientes del sistema. Como se mencionó anteriormente, los nodos de elasticsearch por defecto emplean el puerto 9300 para comunicarse entre ellos por TCP. Por defecto, el puerto está bloqueado por el *firewalld* y no permite ninguna comunicación con otros hosts, por lo que se debe añadir una exclusión abriendo el puerto 9300 en cada nodo, mediante el comando `firewall-cmd --zone=public --add-port=9300/tcp --permanent` y recargando el servicio *firewalld*.

3.1.2. Logstash

En esta breve sección se describirán los aspectos generales de Logstash, una pieza esencial en la plataforma, y su arquitectura en el entrono desplegado. En la sección 3.4 y en el capítulo 4 se explica con más detalle su configuración para conseguir la integración con las fuentes de datos definidas.

Logstash es un software encargado de recolectar datos (recibidos en forma de eventos) con capacidad de transformarlos en tiempo real y enviarlos a un destino definido. Cualquier tipo de evento puede ser enriquecido y parseado mediante una amplia gama de *plugins* de entrada [21], filtrado [22], y salida [23]. Permite una gran escalabilidad horizontal y una fuerte sinergia con Elasticsearch y Kibana. Su instalación es similar a la de Elasticsearch, mediante un paquete RPM disponible en la web [24]. En cuanto a la configuración, Logstash tiene dos tipos de ficheros de configuración: los relativos al procesamiento de los eventos (se explicarán con detalle en el capítulo 4) y los destinados a los ajustes generales.

En relación con estos últimos, el principal fichero de configuración del servicio es *logstash.yml*. Aquí se especifican parámetros como la localización de los ficheros de configuración relativos al procesamiento de los eventos, opciones de los logs, parámetros de la cola interna empleada como buffer de los eventos, o el endpoint REST donde se publican diversas métricas de logstash. Un parámetro importante es el *pipeline.workers*, que define el número de hilos que ejecutan las fases *filter* y *output* y cuyo valor por defecto es el número de cores de la máquina. Otros ficheros importantes son *pipelines.yml*, útil cuando se requieren varios canales de procesamiento, y *jvm.options* para controlar la memoria heap (por defecto reserva 1GB). Estos últimos archivos no ha sido necesario modificarlos.

El modelo de ejecución (canal de procesamiento o *pipeline*) se muestra en la figura 3.5. Cada input definido en Logstash se ejecuta en su propio hilo, escribiendo los eventos en una única cola central almacenada en memoria (por defecto) o en disco procedentes de las fuentes de datos. Posteriormente, cada hilo (según los definidos en *pipeline.workers*) recoge un conjunto de los eventos de la cola (por defecto 125, modificable a través de *pipeline.batch.size*), ejecuta los *filter* configurados, y finalmente ejecuta los outputs una vez los eventos han sido parseados (después de haber ejecutado los *filter*) enviándolos a los destinos especificados en los mismos. Se debe tener en cuenta que al emplear la cola en memoria, si el servicio Logstash finaliza de forma insegura se perderán todos los eventos almacenados en la misma. Para prevenir la pérdida de datos, se debe usar la cola persistente en disco mediante el parámetro *queue.type* en *logstash.yml*.

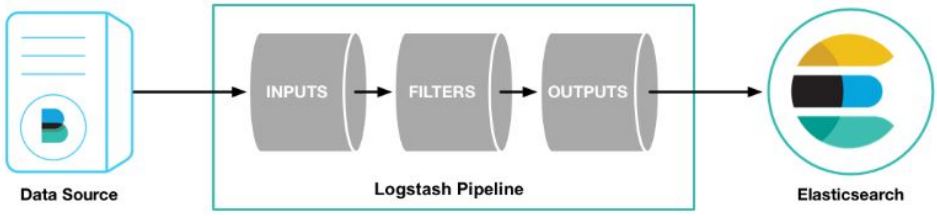


Figura 3.5: *Pipeline* de Logstash. Imagen tomada de [4].

En cuanto a la arquitectura, se han desplegado dos instancias de Logstash, ya que son suficientes para conseguir un rendimiento satisfactorio, siendo posible añadir más en MVs adicionales en caso de no alcanzarlo. Una instancia se ha desplegado en la máquina 1 y otra en la 2 (máquinas de datos) con la misma configuración (se explica la integración de las fuentes de datos y el parseo de los eventos en el siguiente capítulo). De esta manera, como cada una se encuentra en un servidor físico distinto, la redundancia evita la pérdida del servicio ante la caída de uno de ellos.

3.1.3. Kibana

Con respecto a Kibana, como se describe en la sección 2.3.3, es la plataforma de visualización y de análisis diseñada para trabajar con la base de datos Elasticsearch, facilitando la comprensión

de grandes volúmenes de datos. Permite buscar e interactuar con la información almacenada en los índices de Elasticsearch mediante su interfaz gráfica, así como realizar un análisis avanzado de los datos y visualizarlos en una variedad de gráficos como tablas, gráficos de línea de área o de sectores, entre otros. Tanto la instalación de Kibana como su configuración es sencilla.

Del mismo modo que el software anterior, Kibana se ha instalado a través del paquete RPM disponible en la web [25], esta vez en las máquinas con el rol de cliente (3 y 4) encargadas de recibir las peticiones web. Su configuración principal se establece en el fichero *kibana.yml*, por defecto alojado en el directorio `/etc/kibana/`. Aquí es donde se definen importantes parámetros para conseguir la comunicación con Elasticsearch, como su dirección y puerto donde escucha para recibir las peticiones http (por defecto emplea el puerto 9200). En otro parámetro esencial se define la dirección para publicar la aplicación de Kibana, cuyo valor por defecto es *localhost* (IP 127.0.0.1) en el puerto 5601. Al estar publicado en *localhost*, solo se permiten conexiones desde la propia máquina.

Para poder acceder desde otros hosts se establecerá un Nginx como proxy inverso para proteger la aplicación. La instalación de Nginx se ha realizado mediante los repositorios EPEL (Extra Packages for Enterprise Linux). Este servicio se encargará de tomar las solicitudes de clientes, enviarlas al servidor de la aplicación (Kibana), y entregar la respuesta obtenida del servidor al cliente. La principal razón de emplear este proxy inverso para acceder a Kibana es la protección de la aplicación, ya que permite limitar el número de peticiones entrantes y mitigar un ataque de denegación de servicio (DDoS). Otra ventaja esencial de Nginx es su capacidad de cachear el contenido recibido (versiones pre-procesadas de las páginas) de las respuestas de la aplicación y usarlo para responder a los clientes, evitando tener que volver a comunicarse con el servidor de aplicación. Como Kibana se publica en *localhost*, todas las peticiones de hosts remotos deben acceder a través del proxy inverso para conectar con la aplicación. Para conseguir la integración de Nginx con la aplicación, se deben definir una serie de directivas en Nginx; la dirección donde está publicado el servidor de aplicación (mediante `proxy_pass`) y una serie de cabeceras http [26]. Esta configuración principal necesaria de nginx se muestra en el Anexo A.

Además, es necesario abrir el puerto 80 en firewalld y configurar SELinux para evitar problemas de permisos relacionados con scripts y módulos httpd para conectarse a la red. Sin la ejecución de este último comando, Nginx no conseguiría conectarse con Kibana. Una vez aplicado el cambio, teniendo en cuenta el fichero de configuración anterior, Kibana sería accesible a través de la dirección `http://10.10.10.3:80`. Para evitar la pérdida de servicio del sistema, se han instalado dos instancias tanto de Kibana como de Nginx en las máquinas cliente (la 3 y la 4), de modo que se permita seguir accediendo a visualizar los datos y paneles ante la caída de un servidor físico.

3.2. Clúster Keycloak

En este apartado se explica la instalación y configuración de Keycloak, solución empleada para aportar mecanismos de autenticación a la plataforma. Por defecto, la autenticación es local y esto no permite la integración con otros sistemas, por lo que se ha requerido realizar un trabajo extra. De esta forma, se expone cómo se ha conseguido implementar Galera Cluster como base de datos donde Keycloak almacena su información, así como la integración de esta herramienta con Kibana. Lograr estos procesos ha sido relativamente complejo debido a numerosos problemas que ha sido necesario afrontar. A continuación se describe brevemente el trabajo fundamental realizado para conseguirlo, además de la resolución de los principales problemas encontrados de manera concisa.

3.2.1. Instalación Keycloak en modo clúster

Para formar el clúster de Keycloak, se han desplegado dos instancias de Keycloak Server, una en cada máquina cliente (MVs 3 y 4). En cuanto a la instalación, se debe descargar el software desde su página oficial [27], mediante un fichero comprimido que contiene todo lo necesario para desplegar el servidor. Keycloak ofrece 3 modos de operación; el *Standalone*, empleado cuando solo se quiere

desplegar una única instancia de Keycloak server (no recomendable en producción ya que es un único punto de fallo), el *Standalone Clustered*, utilizado para ejecutar Keycloak en modo clúster, y el *Domain Clustered*, también destinado a la tecnología en clúster, pero centralizando la administración de la configuración de los servidores. Se ha escogido este último modo ya que proporciona un lugar central para almacenar y publicar la configuración del clúster, evitando así tener que modificar la configuración en cada uno de los nodos cada vez que se requiera aplicar algún cambio. Esta capacidad está integrada en servidor de aplicaciones WildFly [28], del que se deriva Keycloak.

Para desplegar Keycloak en este modo de ejecución, es necesario conocer los principales conceptos básicos. Primeramente, el *domain controller*, es el proceso responsable de guardar, administrar y publicar la configuración general de cada nodo del clúster, siendo el punto central de donde los nodos obtienen su configuración. El *host controller*, responsable de administrar las instancias del *keycloak server* de una máquina específica (en este caso, solo habrá una instancia en cada máquina). Para reducir el número de procesos en ejecución, un *domain controller* también actúa como un *host controller* en la máquina donde se ejecuta. Por otro lado, el *domain profile* hace referencia a un conjunto de configuraciones usadas por un servidor para arrancar. Se pueden definir múltiples perfiles para consumir por distintos servidores. Por último, el *server group* es un conjunto de servidores administrados y configurados como uno solo, al que puede ser asignado un perfil de dominio (*domain profile*).

En resumen, la configuración del clúster reside en el *domain controller*, iniciado en el nodo máster, y a continuación se ejecuta un *host controller* en cada máquina especificando el número de instancias de Keycloak a iniciar. Estos servidores extraen su configuración del controlador de dominio.

En este caso, se desplegarán dos instancias del servidor Keycloak, una en cada máquina (es decir, una instancia en cada *host controller*), siendo un nodo el máster y otro el esclavo. Se ha establecido la MV 4 como máster del clúster, la cual levanta un *domain controller* y una instancia del servidor Keycloak en modo *domain*. Para conseguir la comunicación con el nodo *slave*, es necesario modificar un fichero de configuración para indicar las interfaces donde escucha el proceso, ya que por defecto se establece en localhost. Este apartado del fichero se muestra en el Anexo B.

De esta forma, Keycloak se publica en todas sus interfaces (0.0.0.0) y puede ser accesible desde cualquier otro host accediendo a su IP y puerto correspondiente de cada servicio. Por defecto, el puerto para la comunicación de los nodos es el 9999, el 9990 es para la gestión del clúster por http, y el 8080 (definido en el `domain.xml`) es para la configuración de general de Keycloak por http y el propio endpoint del servicio de autenticación que ofrece Keycloak. Por lo tanto, es necesario abrir estos puertos en el firewall de CentOS para hacerlos accesibles a otros hosts.

La MV 3 será el *host controller slave*, y requiere la misma configuración que el máster con respecto a las interfaces publicadas. Para iniciarlo, se debe ejecutar el comando `.../bin/domain.sh --host-config=host-slave.xml`. El fichero `host-slave.xml` contiene la configuración para conectarse a un *domain controller* y levanta una instancia del servidor Keycloak. Para conseguir una comunicación segura entre los nodos del clúster y permitir al nodo *slave* obtener su configuración del *domain controller* [29], desde el máster se debe ejecutar el script `.../bin/add-user.sh`, el cual genera un usuario interno administrador y una clave secreta, compartida entre el *master* y el *slave*. Esta clave se debe indicar en el fichero `host-slave.xml`, además de indicar el usuario generado, de la siguiente manera:

```
...
<server-identities>
        <secret value="YWRtX2NpYyE="/>
</server-identities>
...
<domain-controller>
        <remote username="admin-mgmt-cluster" security-realm="ManagementRealm">
            <discovery-options>
```

```
                <static-discovery name="primary" protocol="${jboss.domain.master
                    .protocol:remote}" host="${jboss.domain.master.address:10
                    .10.10.4}" port="${jboss.domain.master.port:9999}"/>
            </discovery-options>
        </remote>
</domain-controller>
...
```

En ambos nodos se ha creado un fichero de servicio para permitir a Keycloak ejecutarse en background e iniciarse automáticamente al encender o reiniciar la máquina virtual. Una vez realizada toda esta configuración, los nodos se comunican correctamente y se consigue iniciar el clúster.

3.2.2. Integración de Galera MariaDB como BBDD

Como se menciona en la sección 2.3.7, con Keycloak viene integrada la base de datos H2 para almacenar su información, pero se recomienda [9] no emplearla en producción ya que no es viable en situaciones de alta concurrencia (por ejemplo, un inicio de sesión paralelo de varios usuarios) y no debe usarse en un clúster. Por ello, se empleará la BBDD MariaDB en clúster (Galera Cluster), más madura con replicación de los datos al instante y topología máster-máster. Se instalará en las MVs cliente 3 y 4.

3.2.2.1. Creación del Clúster MariaDB

En primer lugar, se ha instalado MariaDB (versión 10.3, última estable con Galera Cluster incorporado) en ambas MVs mediante repositorios, y ejecutado el script `mysql_secure_installation` para aumentar la seguridad de la base de datos, configurando aspectos como la contraseña de *root* o eliminar los usuarios anónimos que permiten el acceso sin credenciales. Posteriormente, es necesario crear un usuario con permisos para la sincronización de los nodos, ejecutando los siguientes comandos:

```
mysql -uroot -p -e "CREATE USER sst_user@'%' IDENTIFIED BY 'password-here';"
mysql -uroot -p -e "GRANT ALL PRIVILEGES ON *.* TO sst_user@'%';"
mysql -uroot -p -e "FLUSH PRIVILEGES;"
```

El siguiente paso es configurar MariaDB en cada MV de modo que forme parte de un clúster. Para ello, se ha creado el fichero `cluster.cnf` en el directorio donde MariaDB lee la configuración (`/etc/my.cnf.d/`). En este fichero se deben especificar los parámetros del clúster, principalmente las direcciones IP de los nodos que lo forman, el nombre del clúster, y la IP y nombre del propio nodo. En el nodo de la MV 3 este fichero quedaría de la siguiente manera:

```
[galera]
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0
# Galera Provider Configuration
wsrep_on=ON
wsrep_provider=/usr/lib64/galera/libgalera_smm.so
# Galera Cluster Configuration
wsrep_cluster_name="galera_cluster"
wsrep_cluster_address="gcomm://10.10.10.3,10.10.10.4"
# Galera Synchronization Configuration
wsrep_sst_method=rsync
# Galera Node Configuration
wsrep_node_address="10.10.10.3"
wsrep_node_name="node1-primary-mariadb"
```

En el otro nodo (MV 4), únicamente es necesario modificar los parámetros *wsrep_node_address* y *wsrep_node_name* con sus datos correspondientes. Por último, se deben aplicar una serie de cambios en la seguridad para permitir tanto la correcta comunicación entre los nodos del clúster como el acceso a

la base de datos. Al igual que en ocasiones anteriores, se deben abrir todos los puertos en el firewall de CentOS necesarios para ser accesibles desde otros hosts; estos son el 3306 para conexiones de cliente a mariaDB, 4567 para el tráfico de replicación (tanto TCP como UDP), 4444 para tráfico SST (*State Snapshot Transfer*), empleado para copiar todos los datos cuando se une un nuevo nodo al clúster. Por otro lado, es necesario aplicar una política de seguridad permisiva en SELinux para Galera Cluster. Para ello, instalando el paquete *policycoreutils-python* se puede configurar la política mediante el comando *semanage*. Una vez aplicada la política acorde a la documentación [30], ambos nodos se comunican correctamente formando el clúster.

Es importante indicar la manera de iniciar cada nodo. Cuando se arranca el primer nodo, es decir, no existe ningún clúster, se debe iniciar mediante el script `galera_new_cluster` (disponible al instalar mariaDB con versión superior a la 10.1.8), el cual inicia el servicio de mariaDB con la opción *-wesrep-new-cluster*, de manera que se crea un nuevo clúster. El resto de los nodos, para unirse al mismo, basta con iniciar el servicio de mariaDB con el fichero `cluster.cnf` configurado. En el caso de caída de todos los nodos del clúster, es imprescindible ejecutar `galera_new_cluster` en el último nodo caído para evitar un error al iniciar el servicio, por lo que requiere un reinicio manual. Es posible iniciar el clúster desde un nodo que no haya sido el último en caer, modificando el fichero `/var/lib/mysql/grastate.dat` mediante la línea "`safe to bootstrap: 0`", pero puede causar pérdidas de datos en caso de que se haya insertado nueva información posteriormente a la caída del nuevo nodo creador del clúster.

3.2.2.2. Conexión Galera Cluster con Keycloak

Para establecer Galera Cluster como base de datos donde Keycloak almacene toda su información deben aplicarse ciertas modificaciones en la configuración de Keycloak, tanto en el nodo máster como en el esclavo. En primer lugar, hay que crear la base de datos donde se almacenarán los datos en Galera y un usuario con permisos sobre ésta; se han creado el usuario *keycloak* y la base de datos *keycloak*. Como actualmente no se han creado datos en Keycloak, no hace falta exportar los Realms (información de usuarios, aplicaciones... en Keycloak) a Galera.

Para permitir la conexión de Keycloak con la nueva BBDD, es necesario emplear el conector JDBC de MariaDB, disponible en su web (*mariadb-java-client-2.3.0.jar*) [31]. Posteriormente, crear el directorio `/opt/keycloak/modules/system/layers/base/org/mariadb/main/`, donde se debe alojar el fichero jar, y crear el archivo `module.xml`, indicando la información del driver:

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.3" name="org.mariadb">
    <resources>
        <resource-root path="mariadb-java-client-2.3.0.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

Además, se debe configurar el módulo *datasource* de Keycloak indicando la información de conexión necesaria para permitir la comunicación con la base de datos. Para ello, se deme modificar la sección *datasources* del fichero `domain.xml` de la siguiente manera:

```
...
<datasources>
        <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-name="
            ExampleDS" enabled="true" use-java-context="true">
                <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;
                    DB_CLOSE_ON_EXIT=FALSE</connection-url>
                <driver>h2</driver>
```

```
                <security>
                        <user-name>sa</user-name>
                        <password>sa</password>
                </security>
        </datasource>
        <datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-name="
            KeycloakDS" enabled="true" use-java-context="true">
                <connection-url>jdbc:mysql://localhost:3306/keycloak?useSSL=
                    false</connection-url>
                <driver>mariadb</driver>
                <security>
                        <user-name>keycloak</user-name>
                        <password>keycloak</password>
                </security>
        </datasource>
        <drivers>
                <driver name="h2" module="com.h2database.h2">
                        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
                            datasource-class>
                </driver>
                <driver name="mariadb" module="org.mariadb">
                        <xa-datasource-class>org.mariadb.jdbc.MySQLDataSource</
                            xa-datasource-class>
                </driver>
        </drivers>
</datasources>
...
```

Toda la información para lograr la migración a Galera se ha obtenido de fuentes como GitHub [32][33], la documentación oficial de Keycloak [34], e *issues* publicadas [35]. Una vez aplicada toda esta configuración en los dos nodos, se comprueba que al crear un usuario en una instancia de Keycloak, está disponible también en el otro *Keycloak server*.

3.2.3. Integración de Keycloak con Kibana

El último paso para dotar a la plataforma de mecanismos de autenticación, así como posibilitar su integración LDAP y Directorio Activo, es integrar el clúster de Keycloak desplegado con Kibana. Primeramente, se debe preparar Keycloak para gestionar la autenticación de Kibana configurando un nuevo Realm. En Keycloak, un Realm gestiona un conjunto de usuarios, credenciales, roles y grupos. Cada usuario pertenece y se conecta a un Realm, los cuales están aislados unos de otros y solo pueden administrar y autenticar los usuarios que controlan. Por otro lado, los clientes son entidades que pueden solicitar a Keycloak para autenticar a un usuario. En la mayoría de los casos son aplicaciones o servicios que usan Keycloak para autenticarse y proporcionar una solución *single sign-on*. En este caso, accediendo vía web a la consola de administrador de Keycloak (`http://10.10.10.3:8080/auth/admin/`), se ha creado el Realm *Kibana* y el cliente *kibana* donde se gestionarán tanto los usuarios como políticas de seguridad de la aplicación mediante la interfaz web de Keycloak. La principal modificación en la configuración del nuevo cliente *kibana* en Keycloak es especificar los parámetros *Root URL*, *Valid Redirect URIs*, *Base URL*, *Admin URL* y *Web Origins* de modo que apunten a la dirección del aplicativo. De este modo, se indica la dirección donde redirige Keycloak después de una autenticación, y se devuelven las cabeceras *Access-Control-Allow-Origin* para evitar que el navegador bloquee solicitudes por la política *Cross-origin resource sharing* (CORS) [36].

Inicialmente se intentó realizar la integración de Keycloak con Kibana mediante un plugin disponible en GitHub [37], pero no se logró hacer funcionar debido a la incompatibilidad del plugin (desarrollado para la versión 6.2.4 de Kibana en el momento de la instalación) con la versión de Kibana instalada (6.6.0). Por lo tanto, la integración se ha llevado a cabo a través de la instalación de un proxy desarrollado por CIC, cuyo fichero de configuración creado puede verse en el Anexo C. En este archivo se deben

especificar parámetros como la IP y puerto (se ha definido 127.0.0.1:8081) donde escucha el proxy, la dirección de redirección (en este caso es la aplicación de Kibana, http://127.0.0.1:5601), y la información de Keycloak para la autenticación. En este apartado debe indicarse el Realm, el cliente, la key tanto del Realm como del cliente de Keycloak (obtenidas desde la consola de administrador de Keycloak) y las cabeceras http a reenviar. Se ha creado un fichero de servicio para este proxy en CentOS para su arranque automático, el cual ejecuta el comando `java -jar /opt/keycloak-proxy/bin/launcher.jar /opt/keycloak-proxy/config/proxy.json` para iniciarlo. Este proxy se ha instalado en las dos máquinas cliente, cada una con el fichero de configuración *proxy.json* apuntando a su servidor de Keycloak mediante el parámetro *auth-server-url*. Este cambio en la arquitectura del sistema implica la necesidad de modificar la configuración de Nginx previamente establecida, porque el lugar de redirigir directamente a Kibana, éste debe redireccionar al nuevo proxy instalado, el cual redirige finalmente a Kibana. Por ello, en el Nginx de los dos nodos clientes, debe modificarse el parámetro *proxy_pass* cuyo valor era `http://127.0.0.1:5601` (Kibana) a `127.0.0.1:8081` (proxy de Keycloak).

Como en este caso el acceso a Keycloak se efectúa a través de un proxy inverso o un balanceador, es necesario modificar la configuración tanto de Keycloak como del proxy para permitir a Keycloak conocer la dirección IP origen de la máquina cliente que realiza las peticiones [38]. Keycloak emplea esta dirección en algunas características como SSL, logs, o por si se requiere establecer alguna *whitelist* o *blacklist* de direcciones IP para permitir/denegar peticiones. Para conseguirlo, se debe especificar en el proxy inverso o balanceador la generación de las cabeceras http *X-Forwarded-For* (identifica la IP origen de un cliente) y *X-Forwarded-Proto* (identifica el protocolo origen de una petición http). Actualmente, se deberían definir en Nginx, pero como se explica en la siguiente sección se establecerá un balanceador de carga por delante (HAProxy), por lo que deberán definirse estas *headers* en este nuevo balanceador para mantener la dirección IP real del cliente. Por otra parte, en el lado del servidor de Keycloak, debe especificarse que obtenga la IP cliente de la cabecera *X-Forwarded-For* enviada por el balanceador o proxy, en lugar de tomarla del paquete de red. Para ello, es necesario modificar el fichero `domain.xml` en ambos nodos añadiendo el atributo *proxy-address-forwarding* en el elemento `http-listener`:

```
...
<http-listener name="default" socket-binding="http" redirect-socket="https"
    proxy-address-forwarding="true" enable-http2="true"/>
...
```

Por otra parte, se ha instalado un plugin para establecer un botón de logout permitiendo al usuario cerrar la sesión y volver a la pantalla de login de Keycloak. Este plugin también ha sido desarrollado por CIC, compatible con la versión 6.4.0 de ELK. Se puede instalar simplemente ejecutando el comando `/usr/share/kibana/bin/kibana-plugin install file:./kibana-keycloak-plugin/logout-6.4.0.zip`. Para su funcionamiento, es necesario indicar el endpoint de autenticación de Keycloak, la url Nginx (a donde redirigirá al hacer logout), la cabecera http con el nombre del usuario, y el Realm y cliente definidos en Keycloak para la aplicación, en el fichero principal de configuración de Kibana (`kibana.yml`):

```
logout.keycloakUrl: "http://10.10.10.3:8080"
logout.proxyUrl: "http://10.10.10.3:80"
logout.headerName: "x-forwarded-user"
logout.realm: "Kibana"
logout.client: "kibana"
```

Después de aplicar toda esta configuración en los dos nodos cliente, se comprueba con un nuevo usuario dado de alta en Keycloak que consigue acceder correctamente a Kibana. Sin embargo, al intentar hacer logout se comprueba la existencia de un error, ocasionado por unos cambios producidos a partir de la versión 6.6 de Kibana [39] relacionado con la tramitación de las llamadas API. Siguiendo las instrucciones de los cambios, se modifica el fichero `/usr/share/kibana/plugins/logout/server/routes.js` del plugin de logout implicado para hacerlo funcionar en la versión 6.6.0. En el Anexo D pueden observarse los cambios realizados, permitiendo el correcto cierre de sesión del usuario.

3.3. Clúster HAProxy

Hasta ahora, existen dos servidores Nginx redirigiendo a la página de *login* de Keycloak para posteriormente acceder a Kibana. Esto implica la necesidad de acceder a la IP correspondiente donde está publicado cada Nginx, y en caso de algún fallo en el nodo donde se aloja el Nginx accedido, la imposibilidad de acceso al sistema. Para conseguir un balanceo de carga y tolerancia ante fallos, se desplegará un clúster de HAProxy junto con Keepalived, formado por una instancia en la MV 3 y otra en la MV 4. Como estas máquinas virtuales están hospedadas en servidores físicos distintos, con Keepalived se detectará la disponibilidad de ambos y se conseguirá mantener el servicio de Kibana operativo ante la caída de uno de ellos. Nginx Plus permite la posibilidad de integrar directamente Keepalived con Nginx, pero es una versión de pago por lo que se ha decidido desplegar el balanceador HAProxy por delante. En este apartado describirá brevemente el funcionamiento y la principal configuración de HAProxy y Keepalived, así como su integración con los servidores proxy Nginx.

3.3.1. HAProxy

Este software se encargará de distribuir la carga entre los dos servidores Nginx para mejorar su rendimiento y confiabilidad. La instalación mediante repositorios ya genera un fichero de servicio permitiendo su arranque automático. La configuración definida es la misma tanto para la MV 3 como para la 4, mostrada en el Anexo E. La principal configuración se observa en los apartados *frontend* y *backend* del fichero `haproxy.cfg`. En el *frontend* se especifica la interfaz y puerto donde se publica el HAProxy, es decir, la escucha de todo el tráfico entrante de red a rebalancear, siendo necesario como se explicará en el siguiente apartado con Keepalived, indicar todas las interfaces existentes. Como se pretende establecer el puerto http por defecto (80), es necesario modificar los Nginx de manera que escuchen en un puerto distinto a éste (8085 por ejemplo). Esto implica añadir una exclusión en SELinux ya que no permite levantar este puerto http con Nginx, lográndose con el comando `semanage port -a -t http_port_t -p tcp 8085`, y también se necesita abrir el puerto con *firewalld*. Otro importante aspecto en este apartado es indicar las cabeceras de *forwarding* por el motivo explicado en la sección 3.2.3.

En el apartado de backend es donde se especifica el destino del tráfico (los servidores web Nginx) proveniente del frontend, además del algoritmo de balanceo de carga. Como se puede observar en el Anexo E, se ha definido el algoritmo *Source*, consistente en la obtención de un hash de la IP origen, y en función del peso indicado en los servidores (parámetro *weight*), se designará qué servidor recibirá la petición. De esta manera, la misma IP cliente siempre alcanzará el mismo servidor si está disponible. Se ha especificado distribuir la carga equitativamente indicando el mismo peso para los dos servidores.

3.3.2. Keepalived

HAProxy permite el balanceo de carga entre los servidores Nginx, pero si se cae la propia instancia HAProxy mediante la que se pretende acceder al sistema, implicaría la pérdida del servicio y la imposibilidad de acceso. Para solventar este problema y conseguir tolerancia ante fallo, se empleará el software Keepalived, configurado para comprobar el estado del servicio HAProxy en los dos nodos. Básicamente, su funcionamiento consiste en asignar una dirección IP Virtual a un balanceador definido como máster, la cual se reasignará al otro HAProxy definido como backup en caso de la detección de la caída del máster, convirtiéndose éste en el nuevo maestro. Se ha instalado (mediante repositorios) y configurado Keepalived en las máquinas virtuales 3 (con rol de máster) y 4 (con rol de backup). El principal fichero de configuración para el caso del nodo máster se muestra en el Anexo F. En este archivo, destacan las secciones *vrrp_script* y *vrrp_instance*, donde se especifican los campos necesarios para el protocolo VRRP [40], encargado de asignar la IP virtual. En el apartado *vrrp_script* se indica comprobar si está ejecutándose el proceso *haproxy*, con una frecuencia de 2 segundos. Por otro lado, en la sección *vrrp_instance* se especifica la interfaz empleada para la comunicación con la otra instancia de Keepalived, el estado (máster o backup), un identificador (debe ser el mismo en las dos instancias), un mecanismo de autenticación para la comunicación, la dirección IP virtual a asignar,

además de la propia y la de la otra instancia. Además, la prioridad se empleará para decidir el nodo de asignación de la IP virtual, por lo que el máster debe tener una mayor que el esclavo. En la MV 4 (HAProxy slave) se define la misma configuración del fichero mostrado en el Anexo F, modificando las IPs correspondientes, indicando una prioridad menor, y modificando el parámetro *state* a *BACKUP*.

De esta manera, se permite al usuario acceder a la IP virtual, la cual estará asignada a uno de los dos nodos HAProxy (será el máster por mayor prioridad siempre y cuando esté ejecutándose). Como se comentó anteriormente, es necesario levantar HAProxy en la dirección 0.0.0.0 en lugar de la IP virtual, es decir, en todas las IPs de la máquina virtual. Esto es debido a que el servicio HAProxy genera un error si no se dispone de la dirección IP indicada en su configuración para su publicación en el momento del arranque. A continuación se muestra un esquema de la arquitectura desplegada para acceder a visualizar todos los datos de Elasticsearch mediante Kibana:

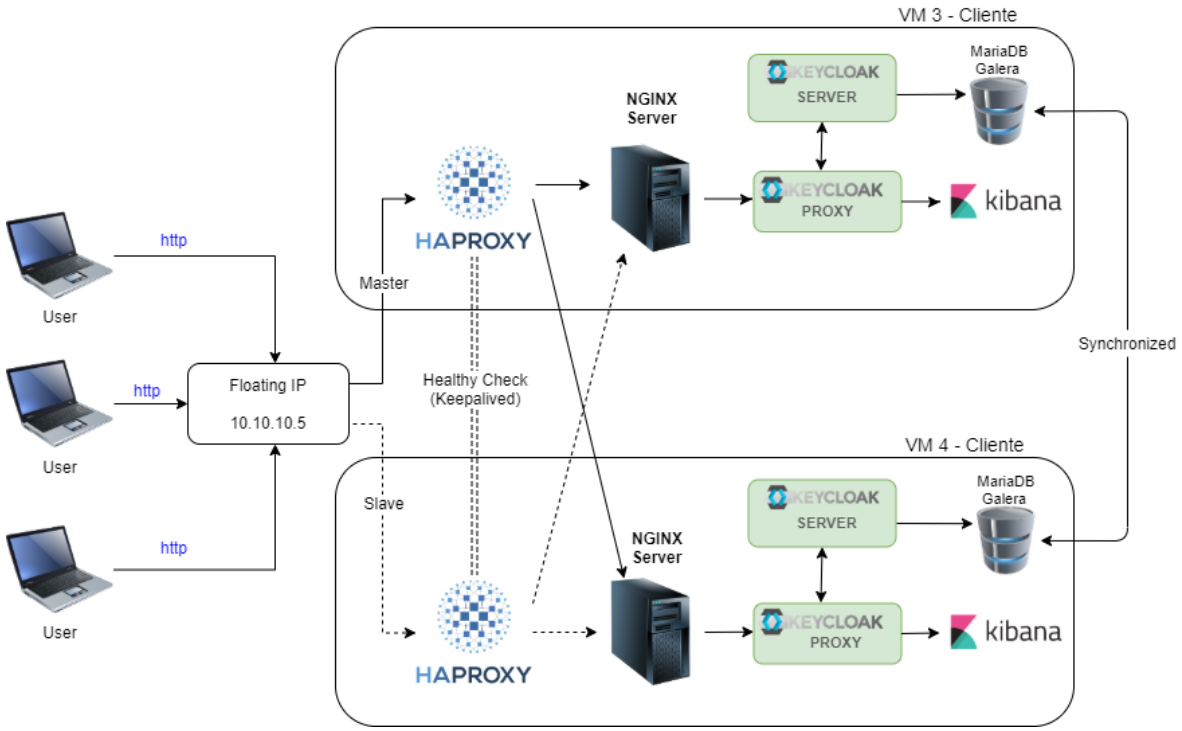


Figura 3.6: Arquitectura del sistema con alta disponibilidad empleando HAProxy con Keepalived.

3.4. Captura de elementos a analizar

En esta sección se describirán los orígenes de datos para analizar en la plataforma, sobre los cuales se aplicarán mecanismos de inteligencia de amenazas para detectar elementos o comportamientos maliciosos, así como enriquecer el dato con más información proveniente de fuentes de inteligencia. También se explicará cómo se integran estos orígenes de datos con el sistema. Se han integrado dos orígenes de datos: tráfico de red de CIC mediante Packetbeat y Suricata, y el IPS (Sistema de Prevención de Intrusos) de un Firewall de CIC.

3.4.1. Tráfico de red

Por un lado, el principal origen de datos será tráfico de red de CIC. En concreto, todo el tráfico que atraviesa el switch de la sala de las gerencias de Sistemas y Comunicaciones, por el que pasa todo el tráfico de los equipos de la sala conectados a la red. Con el objetivo de analizar todo el tráfico de red transcurrido por el switch, el equipo de Comunicaciones ha configurado un puerto del mismo como *port-mirror*, de manera que se envíen copias de todos los paquetes de red recibidos/enviados por el resto de puertos del switch. Esta técnica es muy útil cuando se pretende monitorear el tráfico de red,

así como para IDS (Sistemas de detección de intrusos). Para captar todo el tráfico del puerto espejo, se ha preparado una Raspberry Pi 3b con el sistema operativo Raspbian (distribución de Linux basado en Debian), ya que ha sido necesario conectar directamente el *port-mirror* a un dispositivo hardware capaz de reenviar estos datos a ELK, y así evitar más hardware y/o configuraciones adicionales. Inicialmente se pretendía emplear un mini PC industrial como *sniffer*, con el objetivo de poder emplearlo en un futuro en un entorno industrial. Sin embargo, al final no se pudo disponer del mini PC, por lo que se utilizó la Raspberry Pi en su lugar a pesar de ser menos fiable.

En la RaspberryPi se han configurado dos interfaces físicas (puertos), una de ellas se conecta al *port-mirror* y la otra se conecta a otro puerto del switch, dedicada a IP de gestión para la conexión remota por ssh. Una limitación a tener en cuenta es que la RaspberryPi 3b implementa la interfaz Ethernet a través de un USB 2.0, por lo que ofrece un *thoughtput* máximo de 300Mbps con respecto al puerto gigabit de entrada, y esto puede implicar la pérdida de paquetes en caso de que se supere este ancho de banda. No se estima alcanzar los 300Mbps, pero en ese caso de habría que valorar emplear otros dispositivos con mayor ancho de banda. Los puertos se han conectado mediante dos latiguillos RJ45.

Una vez conectada la Raspberry Pi al puerto espejo, es necesario instalar un software de *sniffing* o analizador de paquetes, de modo que se capture el tráfico transcurrido por la interfaz conectada al *port-mirror* y decodifique los protocolos de red a un formato inteligible. En este caso, se ha empleado Packetbeat, un analizador de paquetes de red en tiempo real ligero, perteneciente a la familia de agentes Beats del propio grupo Elastic, por su fácil integración con Logstash y Elasticsearch. Este software no envía todos los paquetes del tráfico de red en forma de un evento por paquete, sino que decodifica los protocolos, correlaciona peticiones y respuestas, extrae campos (como el tiempo de respuesta) y lo agrupa en un JSON (evento) para su futura indexación en Elasticsearch. Por ejemplo, emplea eventos de tipo flujo donde evento es un grupo de paquetes enviados en un mismo período de tiempo que comparten propiedades comunes, como la misma dirección y protocolo origen y destino.

La Raspberry Pi emplea un procesador con arquitectura ARMv7 de 32 bits, y los agentes Beats oficiales no están compilados para soportar este tipo de arquitectura. Por lo tanto, ha sido necesario realizar un trabajo extra para construir el componente Packetbeat (escrito en el lenguaje Go) mediante una versión de Go compatible con esta arquitectura. La instalación se ha realizado en base a guías encontradas [41], siendo necesaria la descarga tanto del código fuente de Go compatible, como del conjunto de agentes Beats disponible en GitHub [42]. Solo se han instalado Filebeat y Packetbeat, los necesarios empleados.

Después de permitir la correcta ejecución de Packetbeat en la Raspberry Pi, es necesario configurarlo para enviar el tráfico de red transmitido del *port-mirror* a la plataforma ELK. Para ello, en el fichero de configuración principal de Packetbeat (`packetbeat.yml`) deben especificarse ciertos parámetros básicos. En primer lugar, se debe indicar la interfaz donde se pretende capturar el tráfico, siendo en este caso *eth0*, mediante el parámetro `packetbeat.interfaces.device: eth0`. También se ha especificado emplear el método de captura *AF_PACKET* (específico para sistemas Linux) por tener mejor rendimiento que *PCAP* (basado en la librería *libpcap*), este último utilizado por defecto ya que es compatible con la mayoría de plataformas. El último parámetro importante es especificar un *output*, es decir, a dónde se desea enviar el tráfico capturado y decodificado por Packetbeat. En este caso, se enviará a las instancias de Logstash para poder filtrar/modificar los eventos y posteriormente insertarlos en el clúster Elasticsearch. Para ello, se debe especificar el host y el puerto donde está escuchando Logstash en el apartado de Outputs. En este caso, se enviarán al puerto 5043 de las dos instancias de Logstash equilibrando los eventos enviados entre ambas. Para conseguirlo, se han especificado los parámetros: `hosts: ["10.10.10.1:5043", "10.10.10.2:5043"]` y `loadbalance: true` en el fichero `packetbeat.yml`.

Por lo tanto, es necesario configurar Logstash para abrir un socket y escuchar en el puerto 5043

para recibir eventos. En el Anexo G se muestra el fichero de configuración mediante el que Logstash abre un socket en el puerto 5043 a través del plugin *beats* (plugin para recibir eventos de agentes de la familia Beats de Elastic) en su fase *input*, y lo envía a una instancia de Elasticsearch en su fase *output*. El evento se inserta en la base de datos sin ser parseado ni modificado, es decir, de la misma forma en que se recibe. En el capítulo siguiente se mostrará la fase intermedia *filter*, mediante la cual se pretenderá consultar a fuentes de inteligencia, añadiendo su información al propio evento, antes de ser insertado en la base de datos.

Por otra parte, como se explica con más detalle en el siguiente capítulo, se ha integrado Suricata (también desde la RasperryPi) en la plataforma como un sistema de detección de amenazas y de detección de intrusos en tiempo real en base a reglas de diferentes fuentes. Este software emplea su propio módulo para inspeccionar el tráfico de red, también mediante AF_PACKET y detecta y decodifica los protocolos. A diferencia de Packetbeat, Suricata envía los eventos del tráfico de red analizado a un fichero en formato JSON (denominado *eve.json*), el cual se actualiza continuamente. Por lo tanto, para enviar estos datos a la plataforma ELK es necesario un software que lea periódicamente cambios en el fichero y los envíe a Logstash. Filebeat es otro agente de la familia Beats de Elastic que cumple esta función, por lo que se ha instalado de la misma manera que Packetbeat en la Raspberry Pi y configurado para leer el fichero *eve.json* y enviar su información a Logstash (por el puerto 5046). En el Anexo G se observa también la apertura del puerto 5046 para recibir eventos, además de añadir el campo 'origen-evento' con valor 'suricata' al propio evento, con el objetivo de distinguir los eventos provenientes de Suricata y de Packetbeat para posteriormente guardarlos en distintos índices de Elasticsearch.

3.4.2. IPS

Finalmente, una última fuente de datos integrada en el sistema son eventos provenientes de un software IPS instalado en un Firewall de CIC. En base a diversas técnicas de protección, el IPS bloquea los paquetes de red detectados como peligrosos transmitidos en el Firewall. El equipo de comunicaciones de CIC ha configurado el IPS de manera que todos los eventos generados por paquetes de red bloqueados se envíen al puerto 5045 de las instancias de Logstash. El envío de estos eventos se realiza mediante Syslog, un estándar para el envío de mensages de registro en redes informáticas IP, mediante el protocolo UDP por defecto. En el Anexo G se aprecia también la apertura de un socket en el puerto 5045, esta vez mediante el plugin *udp* para recibir paquetes mediante este protocolo.

El objetivo es integrar estos eventos en la plataforma y consultar en tiempo real las fuentes de inteligencia implementadas (se verá en el capítulo siguiente) para conseguir obtener más datos sobre los elementos detectados como amenazas y enriquecer el evento con información útil para los operadores. Esta información se obtiene de las feeds dadas de alta en Hippocampe, y puede contener datos relativos al elemento (url, dominio o IP) detectado como peligroso, como el tipo de ataque que suele emplear, su localización o los servicios que intenta vulnerar, de modo que se pueda ayudar a los operadores a tomar las acciones oportunas de protección.

Capítulo 4

Integración con servicios de inteligencia de amenazas

El presente capítulo explica las soluciones implementadas como mecanismos de ciberseguridad e inteligencia de amenazas, así como sus objetivos, funcionamiento y la integración con ELK. Se emplearán las herramientas Hippocampe para conseguir información de diferentes elementos mediante fuentes de internet, y Suricata para detectar un comportamiento potencialmente peligroso del tráfico de red.

Por un lado, mediante la herramienta Hippocampe se pretende obtener información de IOCs (*Indicator of Compromise* o artefactos maliciosos) de diferentes *feeds* (fuentes) disponibles en internet. Este software admite analizar direcciones IP, URLs y dominios. Su funcionamiento consiste en descargar el contenido de las fuentes dadas de alta en Hippocampe periódicamente e insertar la información en una base de datos Elasticsearch. De este modo, se puede consultar por un observable (IP, URL o domino) vía su interfaz web o API REST y detectar si el elemento existe en la base de datos, y obtener información sobre éste si las fuentes dadas de alta lo proporcionan. El objetivo es emplear Hippocampe para detectar en cada evento proveniente de las fuentes descritas previamente y enviado a ELK, si algún observable es peligroso (si existe en la BBDD de Hippocampe), proporcionar información de éste e incluir estos nuevos datos en el propio evento antes de ser indexado en Elasticsearch para su futura visualización.

Por otro lado, se utiliza el IDS Suricata con el propósito de analizar el tráfico de red y detectar posibles amenazas en base a un conjunto de reglas definidas para distintos protocolos. El principal proveedor que desarrolla servicios para Suricata es *Emerging Threats*, la cual recoge numerosas reglas [43] para los protocolos esenciales que detectan comportamientos anómalos o peligrosos en la red. Mediante Suricata se pretende inspeccionar y analizar el tráfico de red, generando e insertando eventos de tipo alerta en ELK cuando el tráfico coincida con cualquiera de las reglas definidas.

4.1. Hippocampe

La razón principal de escoger Hippocampe es su capacidad de agregar información inmensa disponible en internet de manera local en una base de datos Elasticsearch. Esto permite realizar consultas sobre esta gran cantidad de datos y obtener una respuesta prácticamente de manera instantánea. Esto es muy relevante porque los capturadores de tráfico de red (Packetbeat en este caso) generan un gran número de eventos por segundo, los cuales son enviados a Logstash para ser parseados o modificados, y finalmente indexados en el clúster Elasticsearch desplegado. Otra esencial ventaja es la disposición de una API REST para posibilitar su integración con otros servicios. Por lo tanto, mediante Logstash se pretende tramitar cada evento recibido por Packetbeat de modo que se realice una consulta a Hippocampe mediante su API REST para obtener información sobre uno de los campos del evento (será una IP, URL o dominio), añadir la información proporcionada por Hippocampe en el propio evento, y finalmente insertarlo en Elasticsearch. De esta manera, mediante Kibana pueden visualizarse todos los

eventos del tráfico de red en tiempo real con información sobre la peligrosidad de los IOCs presentes en el evento. El throughput de los eventos generados por Packetbeat inspeccionando el tráfico del *port-mirror* es de unos 2000 por minuto (33 eventos/seg). Además, entre todas las *feeds* dadas de alta en Hippocampe se ha conseguido descargar cerca de un millón y medio de IOCs potencialmente peligrosos. Esto implica realizar 33 consultas por cada elemento requerido por segundo a la base de datos de Hippocampe, averiguando si el elemento existe en la misma. Este throughput es totalmente aceptable por la herramienta gracias al alto rendimiento de Elasticsearch y se consiguen observar los eventos en tiempo real con la información aportada por las fuentes de Hippocampe mediante Kibana. En la figura 4.1 se puede observar la arquitectura del sistema integrado con Hippocampe.

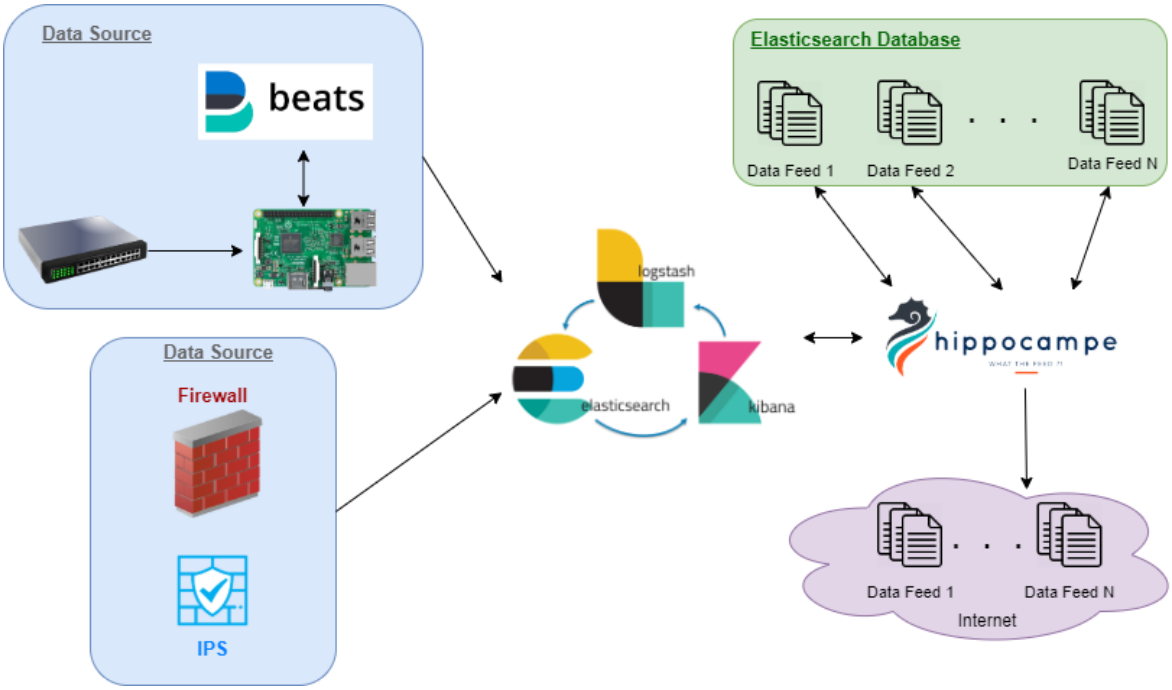


Figura 4.1: Arquitectura del sistema con Hippocampe.

4.1.1. Hipposcore

Para medir el nivel de confianza de un elemento, Hippocampe emplea el concepto de Hipposcore [5], un valor oscilante entre -100 y 100, indicando si es positivo que el observable es probable que sea inofensivo, y negativo si es malicioso, como se muestra en la figura 4.2. De esta forma, cuanto más se acerque al valor -100, mayor es la confianza para considerarse un elemento peligroso, mientras que cuanto más tienda a +100, mayor es la confianza para considerarse un observable no malicioso. Los analistas son capaces de especificar un nivel de confianza a cada fuente dada de alta en Hippocampe en función de la credibilidad o relevancia dada a cada una de ellas. Este valor se emplea para calcular el Hipposcore del observable, además de tener en cuenta el número de *feeds* en las que aparece el observable y el tiempo que lleva el dato en la fuente. Se ha empleado la terminología de la referencia [5] en las ecuaciones descritas. La ecuación 4.3 muestra el cálculo para averiguar el Hipposcore de un elemento, actuando k como un amplificador/atenuador. El valor de P (ecuación 4.2) es el sumatorio del producto de $n1$ (el score definido en la fuente) y $n3$ (edad del dato en la *feed*, fórmula 4.1, donde t es el número de días que lleva el observable en la fuente) de cada fuente dada de alta en Hippocampe. Cuanto mayor sea el número de días que lleva el observable en la *feed*, se considera que su grado de confianza es menos relevante, y en caso contrario, mayor. De esta manera, P es una combinación de la confianza especificada en las fuentes/*feeds* y el tiempo que lleva el elemento en ellas. Si se especifica un score de confianza ($n1$) positivo en la fuente, es decir, se considera que todos los observables de la fuente son inocuos, el Hipposcore será positivo, y negativo en caso contrario.

$$n3 = e^{\frac{-t}{182,625}} \tag{4.1}$$

$$P = \sum n1 * n3 \tag{4.2}$$

$$Hipposcore = [1 - e^{-k|P|}] * \frac{|P|}{P} * 100 \tag{4.3}$$

Figura 4.2: Valor de Hipposcore. Tomada de [5].

4.1.2. Instalación y configuración básica

En relación con la instalación de Hippocampe, se realiza en la MV 3 siguiendo la guía oficial [44], descargando el software desde su repositorio de GitHub. El principal problema encontrado durante este proceso es el requerimiento de Hippocampe de emplear la versión de Elasticsearch 5.1 como base de datos para guardar toda la información de las fuentes. Como la MV 3 actualmente se dispone de una instancia de Elasticsearch 6.6, ha sido necesario desplegar otra instancia de Elasticsearch con la versión 5.1, donde Hippocampe sea capaz de crear el índice de los datos. Como es una instancia independiente de diferente versión, no puede unirse al clúster de Elasticsearch desplegado, por lo que actúa como un clúster de un único nodo. Como Hippocampe solo se ha desplegado en una única MV, se convierte en un punto único de fallo del sistema, por lo que si se cae el nodo donde se hospeda la MV, este servicio no estaría disponible hasta su reinicio. Una alternativa habría sido crear un clúster de Elasticsearch con 3 nuevas MVs dedicadas exclusivamente a Hippocampe, pero no se dispone del suficiente hardware libre requerido en los nodos físicos para hospedarlas. Mediante un fichero de configuración principal se especifica dónde se aloja la instancia de Elasticsearch para crear el índice de Hippocampe y almacenar su información, así como la dirección y puerto para atender las peticiones API y publicar la interfaz web (en este caso se ha indicado 0.0.0.0:5000 para permitir realizar consultar desde otros hosts y mediante localhost).

El proceso de descarga de las fuentes dadas de alta en Hippocampe se denomina *shadowbook*, y puede ejecutarse accediendo a la interfaz web de Hippocampe o mediante una llamada a su API, por ejemplo, con el comando *curl* (`curl -XGET localhost:5000/hippocampe/api/v1.0/shadowbook`). Cuando se lanza, este servicio recoge todos los datos de cada *feed* a través de internet y los indexa en el índice de Elasticsearch. Para mantener la información en la base de datos local actualizada con respecto a las fuentes dadas de alta, este proceso debe ejecutarse regularmente. Para ello, se ha programado una tarea periódica a través del demonio Cron de Linux, de modo que se ejecuta un script sh creado, el cual realiza la llamada API al *shadowbook*. Se ha programado una ejecución diaria a las 4 de la mañana indicando lo siguiente en el fichero Crontab (minuto, hora, día del mes, mes, día de la semana): `00 4 * * * /opt/hippocampe/scripts/updateFeeds.sh`.

4.1.3. Feeds de Hippocampe

Para dar de alta fuentes en Hippocampe, se debe crear un fichero de configuración asociado a cada una de ellas, en el directorio `.../hippocampe/core/conf/feeds/`. Entre otras cosas, en este archivo se especifica la dirección url donde está publicada la *feed*, el nombre de todos los campos de la fuente (puede tener una ip, una descripción, un fecha...), el carácter que delimita o separa cada campo, así como el tipo de cada uno (string, fecha, entero...) para su indexación en Elasticsearch, el número de línea donde comienzan a citarse los elementos, y el score asociado a la fuente en función de una confianza dada. En el software de Hippocampe están incluidas varias fuentes con sus ficheros de configuración; sin embargo, actualmente algunas de ellas no funcionan correctamente. Esto es debido a que a día de hoy la última actualización de los ficheros de configuración de las fuentes en Hippocampe

se ha realizado en 2.018. Por lo tanto, desde esa fecha hasta la actualidad,las fuentes pueden haber sido modificadas de alguna manera afectando a la integración con Hippocampe, por ejemplo, porque han dejado de ser mantenidas y la *feed* ya no está publicada, porque se ha cambiado su licencia y no está disponible de manera libre, porque se haya migrado a otra dirección URL, o porque se han modificado los campos presentes en la fuente. Estos dos últimos casos pueden resolverse modificando el fichero de configuración asociado a la fuente, pero si ésta ya no se publica, se debe deshabilitar eliminando el fichero de configuración.

Los ficheros de configuración de las fuentes integradas en el software base de Hippocampe están disponibles en su repositorio de GitHub [45]. La integración con varias de estas fuentes fallan debido a modificación reciente de las fuentes, por lo que se ha realizado un proceso de modificación de estos ficheros para su correcta integración. Además, se han deshabilitado las que han dejado de publicarse. También se ha integrado alguna fuente nueva, obtenidas de la página web `https://threatfeeds.io/`, donde de manera actualizada publican nuevas *feeds* gratuitas. Hay que tener en cuenta que las fuentes deben ser accesibles de manera libre y gratuita, ya que existen fuentes con otro tipo de licencia, por ejemplo, con requerimiento de pagar una subscripción para adquirir el servicio. En un entorno de producción sería relevante valorar el uso de este tipo de feeds, ya que se garantizaría una mantenibilidad con actualizaciones y disponibilidad de la fuente. Después de haber añadido alguna *feed* nueva, eliminado las no disponibles y corregido diversos ficheros de configuración, en el Anexo H se muestran todas las URLs de las fuentes integradas correctamente en Hippocampe actualmente. A día de hoy, todas estas fuentes componen un total de 178.867 dominios, 588.983 direcciones ip y 18.761 direcciones URL, teniendo en cuenta la posibilidad de elementos repetidos entre distintas fuentes. Estos números se actualizarán cada vez que se lance el proceso *shadowbook* en caso de que se añadan/eliminen elementos en las *feeds*.

Una de las nuevas fuentes integradas más interesante es la de *StevenBlcak/hosts* (información en `https://github.com/StevenBlack/hosts`), presentando numerosos dominios de carácter maligno y de categorías inapropiadas unidos en un único fichero. Esta fuente resulta ser particularmente complicada de incorporar porque sus campos están separados por un espacio (ver fuente en el Anexo H), e Hippocampe no permite indicar un espacio como el carácter delimitador en el fichero de configuración de la fuente. Para conseguirlo, se ha modificado el código python del software de modo que permita especificar un carácter en el fichero de configuración de la fuente e Hippocampe entienda que el carácter que separa cada campo de la fuente es un espacio. Hippocampe emplea la clase de `DictReader` del módulo `csv` de Python para realizar el parseo (identificación de campos) de la *feed*. Uno de los parámetros que recibe esta clase es el elemento separador o delimitador de los campos. A continuación se muestra un cuadro con el código modificado del fichero `.../hippocampe/core/services/modules/shadowbook/parser.py`, donde ejecuta la acción descrita. El valor de la variable *delimiter* es el carácter especificado en el fichero de configuración de la fuente (asignado previamente). Por lo tanto, se ha añadido el primer caso *if*, de manera que tramite el carácter '\s' indicado en el fichero de configuración de la fuente, como un espacio, definiendo el argumento ´delimeter' del objeto de la clase *DictReader* con un espacio. El segundo caso *elif* simplemente envía como argumento el carácter delimitador especificado en el fichero de configuración de la fuente al objeto *DictReader* para su parseo. El último *if* permite indicar el delimitador '\t' en el fichero de configuración de la fuente para parsear la fuente con un tabulador como carácter delimitador de los campos.

```
...
        if delimiter == '\\s':
                parsedPage = csv.DictReader(parsedPage, fields,
                   extraFields, delimiter = ' ')
        elif delimiter != '\\t':
                parsedPage = csv.DictReader(parsedPage, fields,
                   extraFields, delimiter = delimiter)
```

```
        if delimiter == '\\t':
                parsedPage = csv.DictReader(parsedPage, fields,
                   extraFields, dialect = 'excel-tab')
        return parsedPage
...
```

4.1.4. Parseo con Logstash

Como se describe en la sección 3.4, mediante Logstash se recibe y envía cada evento, con las fases *input* y *output* respectivamente, de los orígenes a la base de datos Elasticsearch. La fase *filter* es una intermedia encargada de la modificación del evento, a través de numerosos plugins disponibles. Este proceso es el más complejo de explicar y llevar a cabo debido a sus numerosos detalles y particularidades, por lo que se describirán las principales funciones que se realizan en esta fase.

La fase *filter* permite ejecutar código Ruby para modificar los campos del evento, por ejemplo, indicando un script mediante el plugin *ruby* de Logstash. Aprovechando esta funcionalidad, se ha creado un script (disponible en el Anexo I) en este lenguaje para consultar en cada evento recibido, a Hippocampe por el elemento deseado e integrar esta respuesta al propio evento. Como se ha comentado anteriormente, Hippocampe expone una API REST con una serie de servicios. Uno de ellos es el *more*, el cual devuelve información de un elemento en base a todos los datos presentes en las fuentes integradas, como su hipposcore o las fuentes que contienen este elemento. Es necesario indicar en la llamada el propio elemento y su tipo (ip, url o dominio). A modo de ejemplo, se muestra una petición al servicio *more* consultando la dirección IP 185.176.27.2 y la respuesta de Hippocampe:

Query

```
curl -i -H "Content-Type: application/json" -X POST -d '
{
        "185.176.27.2" : {"type" : "ip"}
}' http://10.10.10.3:5000/hippocampe/api/v1.0/more
```

Response

```
{
   "185.176.27.2":[
      {
         "extra":[
            "4",
            "2",
            "Malicious Host",
            "RU",
            "",
            "55.7386016846,37.6068000793",
            "3"
         ],
         "firstAppearance":"20200416T103747+0200",
         "hipposcore":{
            "hipposcore":"-99.61"
         },
         "idSource":"AXGCIYcNL-T_vBY97ud7",
         "ip":"185.176.27.2",
         "lastAppearance":"20200416T103747+0200",
         "lastQuery":"20200501T040320+0200",
         "source":"http://reputation.alienvault.com/reputation.data"
      },
      {
         "firstAppearance":"20200416T103336+0200",
         "hipposcore":{
            "hipposcore":"-99.61"
         },
```

```
            "idSource":"AXGCHy0rL-T_vBY96seQ",
            "ip":"185.176.27.2",
            "lastAppearance":"20200416T103336+0200",
            "lastQuery":"20200501T040015+0200",
            "source":"http://blocklist.greensnow.co/greensnow.txt"
         },
         {
            "firstAppearance":"20200419T040206+0200",
            "hipposcore":{
               "hipposcore":"-99.61"
            },
            "idSource":"AXGCIT2OL-T_vBY97mcn",
            "ip":"185.176.27.2",
            "lastAppearance":"20200419T040206+0200",
            "lastQuery":"20200501T040150+0200",
            "source":"http://cinsscore.com/list/ci-badguys.txt"
         }
      ]
}
```


En este caso, el elemento está en dos de las fuentes integradas en Hippocampe, con un *hipposcore* de -99.61, por lo que se trata de un observable malicioso. Si el elemento no existe en ninguna fuente, la respuesta es un array vacío. En el script elaborado se realiza una consulta al servicio *more* por cada evento recibido en función de su tipo. Entre otros, Packetbeat genera eventos del tráfico analizado de tipo flow (grupo de paquetes enviados en el mismo periodo de tiempo con propiedades comunes, como mismas IPs orígenes y destinos), de tipo http (como peticiones web) o dns (solicitudes dns), por lo que cada tipo de evento presenta diferentes campos. En el caso de los eventos recibidos por Packetbeat, se consulta por el campo de la IP destino en los eventos de tipo flow, en los eventos dns por el dominio solicitado y en los http por la dirección URL solicitada. La respuesta de Hippocampe se inserta en un campo nuevo del propio evento.

Al igual que las fases de *input* y *output* de Logstash, la fase *filter* también se define mediante uno o varios ficheros de configuración. Para el caso de los eventos recibidos por Packeteat, el fichero creado se puede ver en el Anexo J. Mediante este fichero se invoca al script previamente mencionado y se realizan una serie de parseos para modificar o crear campos en el evento. Alguno de los parseos realizados, por ejemplo, se encargan de modificar el nombre de los campos, y de crear un nuevos campos en función de la respuesta obtenida en formato JSON de Hippocampe. De esta manera, se consigue crear campos en el evento con los distintos campos de la respuesta de Hippocampe.

En relación con los eventos procedentes del sistema IPS, se ha creado otro script similar, adaptado al formato del evento recibido y consultando por el elemento de la dirección IP origen de todos los eventos recibidos (paquetes bloqueados por el IPS). También se ha creado otro fichero de configuración de Logstash, adaptado y similar al que se muestra en el Anexo J.

4.2. Suricata

El objetivo de esta sección es explicar la configuración básica de Suricata para su funcionamiento en este entorno, las reglas aplicadas y su integración con ELK. Como se menciona en el apartado 3.4, Suricata dispone de un módulo para realizar *sniffing* y produce una salida del tráfico analizado en un fichero en formato JSON (*eve.json*). Durante este proceso de análisis, Suricata decodifica los protocolos de alto nivel y procesa el tráfico comparándolo con las reglas integradas, generando un evento de tipo alerta e incluyéndolo en el fichero *eve.json* en caso de cumplimiento de alguna de ellas. Una limitación de esta herramienta es que no permite descifrar el tráfico, por lo que solo puede escanear la parte no cifrada (encabezados) de las conexiones que empleen TLS como HTTPS, pero no el *payload* en sí. Mediante el agente Filebeat se lee este fichero y se envía su contenido a Logstash por cada evento generado por Suricata. Posteriormente desde Kibana se pueden realizar filtros para visualizar

únicamente los eventos de tipo alerta, como se ve en el capítulo 5.

Suricata se ha instalado en la RaspberryPi de manera sencilla a través de los repositorios incluidos en el sistema operativo, para procesar todo el tráfico en la interfaz configurada como *port-mirror*. Su configuración principal reside el fichero *suricata.yaml*. Entre otros parámetros, aquí se especifica la interfaz cuyo tráfico se desea inspeccionar, el tipo de logs a producir, las reglas habilitadas o los protocolos a analizar. Además, permite definir una subred como tu red local (con el parámetro HOME_NET) y así lograr distinguir si una dirección IP pertenece a la red local o a una red externa, ya que muchas reglas emplean la dirección de la comunicación para generar alertas.

4.2.1. Reglas de Suricata

Una vez se ha instalado y realizado la configuración básica de Suricata, se deben definir las reglas a emplear para comparar el tráfico procesado con éstas y generar alertas si ocurre un *match* con alguna de ellas. El software base de Suricata ya integra numerosas reglas (`https://github.com/OISF/suricata/tree/master/rules`) para protocolos como http, dns o dhcp entre otros. Además, incluye la funcionalidad `suricata-update`, permitiendo gestionar e introducir más conjuntos de reglas existentes elaboradas por diversas organizaciones para Suricata. Algunas de estas fuentes requieren suscripción de pago y otras son abiertas. La fuente gratuita más utilizada es *Emerging Threats*, un centro de investigación de seguridad abierta dedicado a producir fuentes de datos sobre amenazas nuevas y actualizadas para plataformas IDS/IPS como Snort o Sutricata. Mediante la ejecución del comando `suricata-update`, se integra directamente el conjunto de relgas de la compañía *Emerging Threats*, y las actualiza si ya están habilitadas. Este conjunto de reglas analizan diversos protocolos, comprueban comportamientos anómalos o sospechosos en la red, y detectan de numerosos elementos maliciosos como direcciones IP. Todas las reglas pueden visualizarse en [43]. Además, Suricata permite crear manualmente nuevas reglas para poder generar alertas sobre algún tipo de tráfico en concreto que pueda resultar útil en la organización. Las reglas tienen un formato específico, y se componen de tres apartados:

- Acción: determina lo que ocurre cuando la regla coincide. Existen cuatro tipos; *Pass* para dejar de escanear el paquete ignorando las reglas, *Drop* para eliminar o bloquear el paquete (solo disponible cuando Suricata se ejecuta en modo IPS), *Reject*, para enviar un paquete de rechazo de la comunicación tanto al remitente como receptor (con modo IPS activado el paquete también se *dropea*), y *Alert* para generar un evento de tipo alerta. La acción *Alert* es la que se empleará siempre en este caso, ya que al proceder el tráfico de un *port-mirror*, no tiene sentido hacer *drop* de paquetes.

- Cabecera: Define el protocolo, direcciones IP, puertos, y dirección de la comunicación. Permite numerosos protocolos, como por ejemplo tcp, udp, icmp, ip, http, ftp, smb, dns, modbus, nfs, ntp, smtp, krb5...

- Opciones de regla: Especifica los detalles de la regla. Aquí es donde se indican condiciones detalladas mediante el uso de palabras claves propias de cada protocolo. Por ejemplo, se podría definir si existe un determinado string en el cuerpo de una petición del protocolo http, o definir un ttl del paquete en el protocolo ip. Existen numerosas combinaciones y posibles atributos que permiten a los expertos en este área crear reglas muy específicas para evitar o detectar ciberataques.

A continuación se muestra un ejemplo de una regla muy sencilla que ha coincidido con el tráfico inspeccionado por Suricata, generando una alerta:

alert tcp any any -> any ![80,8080] (msg:"SURICATA HTTP but not tcp port 80, 8080"; flow:to_server; app-layer-protocol:http; sid:2271001; rev:1;)

En esta regla se define realizar la acción de generar una alerta cuando se detecta un paquete tcp en una petición http, desde cualquier dirección ip y puerto orígenes, hacia cualquier dirección ip a un puerto destino distinto de 80 y 8080. Suricata detecta este tráfico como anómalo porque los puertos destino no son los estándar para realizar peticiones a un servidor web mediante http.

4.2.2. Parseo con Logstash

Como se menciona en la sección 3.4, la información generada por Suricata se envía mediante Filebeat a las instancias de Logstash para parsear los eventos y almacenarlos en Elasticsearch. La fase *filter* realizada para este parseo es sencilla ya que el mensaje se recibe en un formato JSON. Logstash dispone de un plugin para tramitar este formato y desacoplar todos los campos del mensaje JSON creando un campo por cada uno de ellos en el evento. Simplemente es necesario indicar cuál es el campo del evento donde se encuentra el mensaje (por defecto, la información enviada a Logstash se almacena en un campo denominado *message*). De esta manera, se crea el fichero de configuración de Logstash para parsear los eventos de Suricata:

```
filter {
  if [origen-evento] == "suricata" {
    json {
      source => "message"
    }
  }
}
```

4.3. Alertas

Una vez integrados estos servicios para detectar posibles ciberamenazas, se ha elaborado un mecanismo de generación de notificaciones o alertas mediante el software ElastAlert. Este framework es desarrollado por la comunidad, y como se describe en la sección 2.3.5, está específicamente creado para su integración con Elasticsearch y producir alertas cuando coincidan los patrones especificados. El software se ha descargado e instalado siguiendo las instrucciones de su repositorio en GitHub [46] en la MV 4.

El principal funcionamiento de ElastAlert consiste en crear tipos de reglas definiendo los patrones convenientes, y especificar la alerta o acción a realizar cuando se cumpla el patrón. Elasticsearch es consultado periódicamente y sus datos se envían al tipo de regla, para determinar si ésta se cumple o no se cumple. Si coincide, se lleva a cabo la alerta definida para la regla.

Respecto a su principal configuración, reside en el fichero `config.yaml`, donde se establecen parámetros como la frecuencia con la que ElastAlert realiza las consultas a Elasticsearch o el directorio donde se alojan los ficheros de configuración de las reglas implementadas. Cada regla se define en uno de estos ficheros. En este caso, se ha creado una regla para comprobar si existe el campo "hipposcore" en algún evento en Elasticsearch. Dicho campo se crea durante la fase de parseo con Logstash sólo si Hippocampe detecta el elemento consultado (ip, dominio o url) en alguna de sus fuentes. En caso afirmativo, se genera una alerta de tipo Email, de manera que se notifique vía correo electrónico la detección del elemento en Hippocampe.

Para realizar esta acción, se crea el siguiente fichero de configuración:

```
# Regla que envia un email cuando existe el campo hipposcore
es_host: 10.10.10.4
es_port: 9200
name: Dangerous Element
index: packet*
```

```
type: any
realert:
  minutes: 0
filter:
 - query_string:
      query: "_exists_:hipposcore"
alert:
- "email"
email:
- "asbarcena@cic.es"
smtp_host: "10.10.10.35"
smtp_port: 25
#smtp_ssl: true
from_addr: "elastalert@cic.es"
alert_text: "Se ha detectado el acceso a un elemento peligroso de tipo {0} a las
    {1}.

URL del Dashboard: http://10.10.10.5

Informacion  detallada del evento:"
alert_text_args: ["type", "@timestamp"]
```

Como se puede observar, es necesario indicar diversos parámetros. Por un lado, la dirección y puerto donde se encuentra la instancia de Elasticsearch y el índice donde se almacenan los eventos sobre los que se consultan. En este caso, los eventos recibidos por Packetbeat se almacenan en índices diarios cuyo nombre siempre comienza por *packet*, por lo que la regla verificará todos los eventos recibidos del tráfico analizado por Packetbeat. El tipo de la regla es *any* para generar la alerta siempre que la consulta devuelva algún evento. También se puede especificar un periodo de tiempo mediante el parámetro *realert*, durante el cual no se generan alertas repetidas, pero en este caso no se ha habilitado. El último parámetro indicado es el *filter*, a través del cual se realiza la consulta deseada utilizando el lenguaje Query DSL. En este caso, se consulta por todos los eventos que tengan un campo llamado "hipposcore". De esta manera, cada minuto ElastAlert consulta en los índices que comiencen por *packet* todos los nuevos eventos almacenados en Elasticsearch, y genera una alerta por cada uno que tenga el campo "hipposcore".

Una vez definida la regla, se debe especificar la alerta o acción a realizar; en este caso, una alerta de tipo Email. Esta alerta requiere especificar el servidor de correo y puerto SMTP (se deberá permitir la comunicación entre la máquina de ElastAlert y el servidor SMTP por el puerto 25) para el envío de los correos. También debe indicarse la dirección origen y el destinatario. Además, permite indicar el asunto y cuerpo del correo, con los parámetros *name* y *alert_text*, respectivamente. Al final del mensaje del correo también se envía la información de todos los campos del evento (documento JSON) que ha generado la alarma. A continuación se muestran tres capturas con el ejemplo de un email recibido por este tipo de alerta, disparada por la detección de un dominio malicioso (*vms.boldchat.com*) en el tráfico de red analizado por Packetbeat en una de las fuentes de Hippocampe:

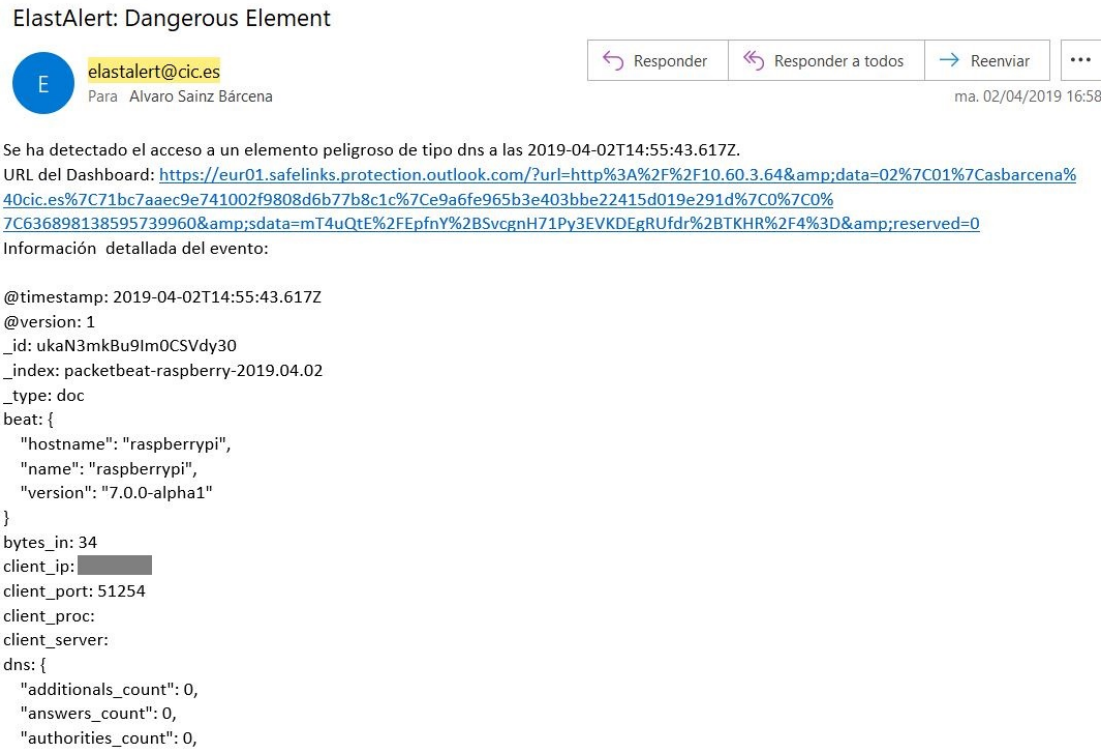


Figura 4.3: Primera parte del Email generado por ElastAlert.

Figura 4.4: Segunda parte del Email generado por ElastAlert.

hippocampe: {"vms.boldchat.com":[{"domain":"vms.boldchat.com","firstAppearance":"20190313T110736+0100","hipposcore":{"hipposcore":"-83.35"},"idSource":"AWl2g1WVARN9n6IBZIbT","lastAppearance":"20190313T110736+0100","localhost":"127.0.0.1","source":"https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fhosts-file.net%2Femd.txt&data=02%7C01%7Casbarcena%40cic.es%7C71bc7aaec9e741002f9808d6b77b8c1c%7Ce9a6fe965b3e403bbe22415d019e291d%7C0%7C0%7C636898138595739960&sdata=2nLTTaE1FvFbNEIqLd3YvqhKEM3YtDxWzXUs2v0G8DU%3D&reserved=0"}]}

hipposcore: hipposcore:-83.35
hipposcoreFinal: -83.35
host: {
  "name": "raspberrypi"
}
idSource: AWl2g1WVARN9n6IBZIbT
ip: 10.60.2.35
lastAppearance: 20190313T110736+0100
localhost: 127.0.0.1
method: QUERY
notes: No response to this query was received
num_hits: 2
num_matches: 2
port: 53
proc:
query: class IN, type A, vms.boldchat.com.
resource: vms.boldchat.com.
server:
status: Error
tags: [
  "beats_input_raw_event"
]
transport: udp
type: dns
urlSource: https://eur01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fhosts-file.net%2Femd.txt&data=02%7C01%7Casbarcena%40cic.es%7C71bc7aaec9e741002f9808d6b77b8c1c%7Ce9a6fe965b3e403bbe22415d019e291d%7C0%7C0%7C636898138595749969&sdata=nT92YbECx3so5uQcdsNsm%2BxzvUn25mxhwV8pN0ejsdw%3D&reserved=0

Figura 4.5: Última parte del Email generado por ElastAlert.

Capítulo 5

Resultados

En este capítulo se muestran las visualizaciones y paneles (*dashboards*) elaborados con Kibana de todos los datos integrados en la plataforma. Mediante esta herramienta, los operadores son capaces de realizar análisis y diversas búsquedas rápidamente sobre todos los datos (ya parseados con Logstash) indexados en Elasticsearch, además de visualizar la información en gráficos de forma centralizada.

Se ha creado un panel para cada fuente de datos compuesto por un conjunto de diversos tipos de visualizaciones o gráficos, permitiendo a los operadores obtener una visión general del tráfico de red y visualizar las alarmas, sus causas y elementos implicados para actuar en consecuencia. A continuación se exponen los *dashboards* creados para cada fuente. Por motivos de confidencialidad y seguridad, se han ocultado en las imágenes todos los datos relativos al direccionamiento IP privado de la red de CIC.

Por otro lado, no se han podido realizar pruebas de carga sobre la plataforma en producción, pero se ha verificado que el sistema acepta todo el tráfico real atravesado por la red. Además, se ha comprobado que todos los eventos recibidos son comparados con todas las fuentes dadas de alta en Hippocampe, con un total de casi un millón y medio de elementos peligrosos.

5.1. Dashboard Tráfico de red: Packetbeat con Hippocampe

En primer lugar, el panel elaborado para los datos recibidos por Packetbeat del tráfico de red, se muestra en las figuras 5.1 y 5.2. La primera muestra un histograma con el número de eventos totales recibidos y otro con el número de eventos cuyo observable consultado a Hippocampe se considera una amenaza, es decir, existe en alguna de las *feeds* dadas de alta. En las siguientes filas se han creado dos diagramas de sectores, uno con el top de direcciones IP origen más repetidas (mayor número de eventos) y otro con el top de IPs destino. Además del histograma, se han elaborado visualizaciones con el número de eventos tanto por los todos los recibidos por Packetbeat para cada evento de tipo flow, dns o http como los que han generado una amenaza en Hippocampe. Estas visualizaciones se han creado de manera que el usuario puede hacer click en una de ellas y se aplica un filtro en Kibana para mostrar únicamente dicho tipo de eventos en todo el panel actualizado. De esta manera el operador puede filtrar y observar el evento que ha causado una amenaza rápidamente. En este caso, se ha establecido un intervalo de tiempo de 24 horas (modificable), durante en cual se han analizado 1.056.492 de eventos de tipo flow, 6.399 de dns, 825 de http, y no se ha detectado ninguna amenaza con Hippocampe.

En la segunda imagen (figura 5.2) se muestra un tipo de visualización llamada búsqueda, donde se puede observar todo el detalle de los eventos. Se pueden añadir todos los campos necesarios a la visualización, y desplegando un evento en concreto se detallan todos sus campos.

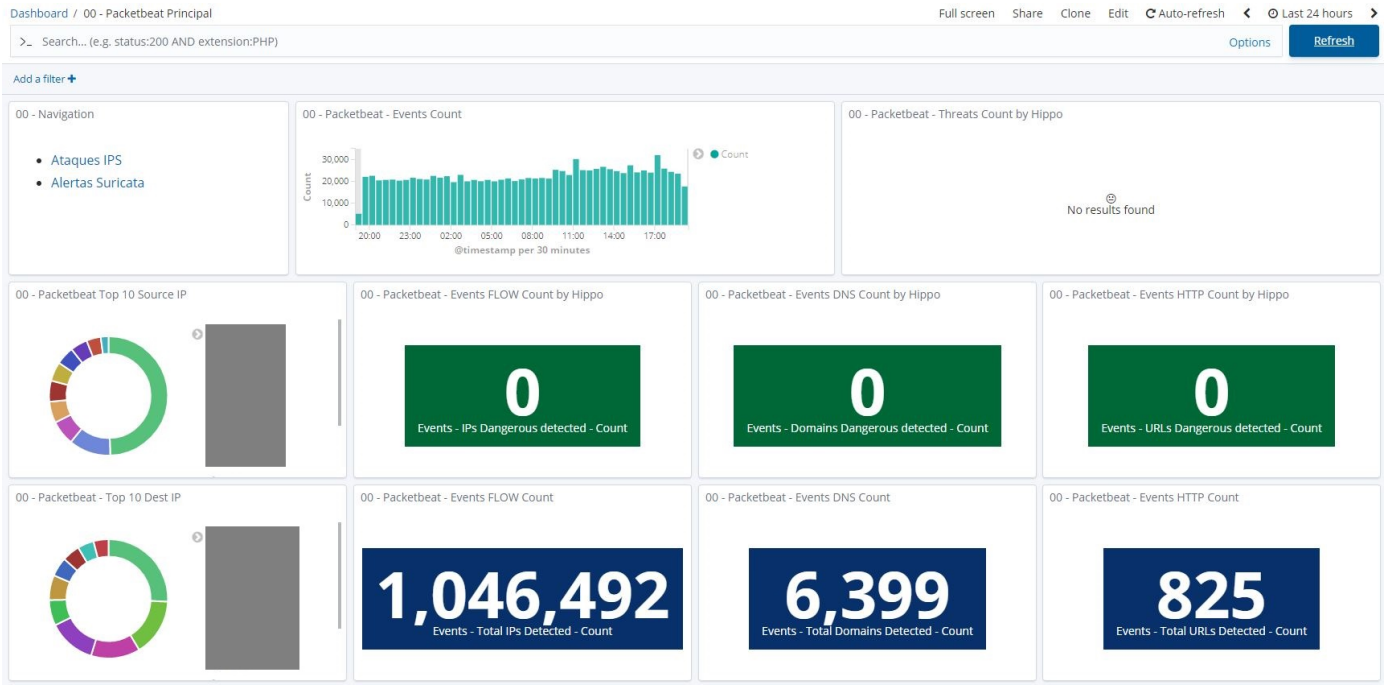


Figura 5.1: Dashboard Packetbeat con Hippocampe con el número de eventos recibidos y amenazas detectadas.

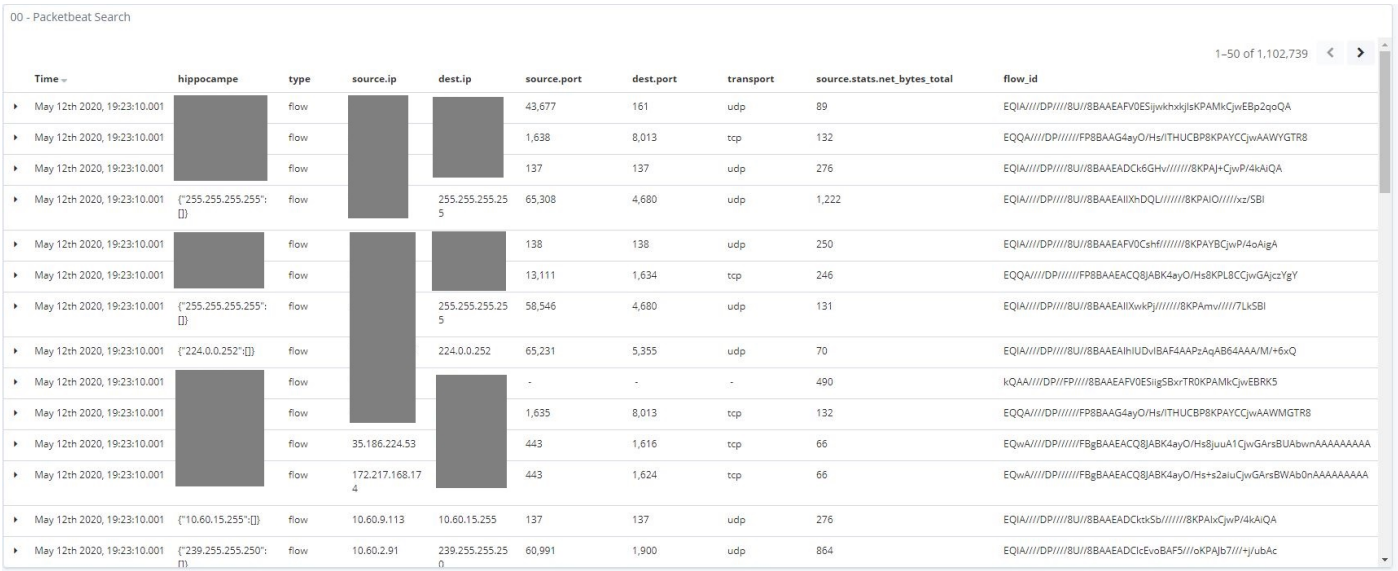


Figura 5.2: Dashboard Packetbeat con Hippocampe con el detalle de cada evento. Se censura la información que contiene detalles internos de la red.

A modo de ejemplo, en las figuras 5.3, 5.4, 5.5 y 5.6 se muestra el panel filtrando por eventos dns. Como se puede apreciar, se actualiza todo el *dashboard* de modo que únicamente se aprecian los eventos cuyo campo *type* es dns. Es posible aplicar tantos filtros como se deseen para realizar consultas más complejas cobre todos los datos. En las imágenes 5.4, 5.5 y 5.6 se puede visualizar el detalle de un evento dns desplegado con todos sus campos. Se puede comprobar que es una petición dns al dominio *3.debian.pool.ntp.org*, y la consulta a hippocampe no devuelve información (campo *hippocampe*), por lo que no existe en ninguna de sus *feeds* y no se considera una amenaza.

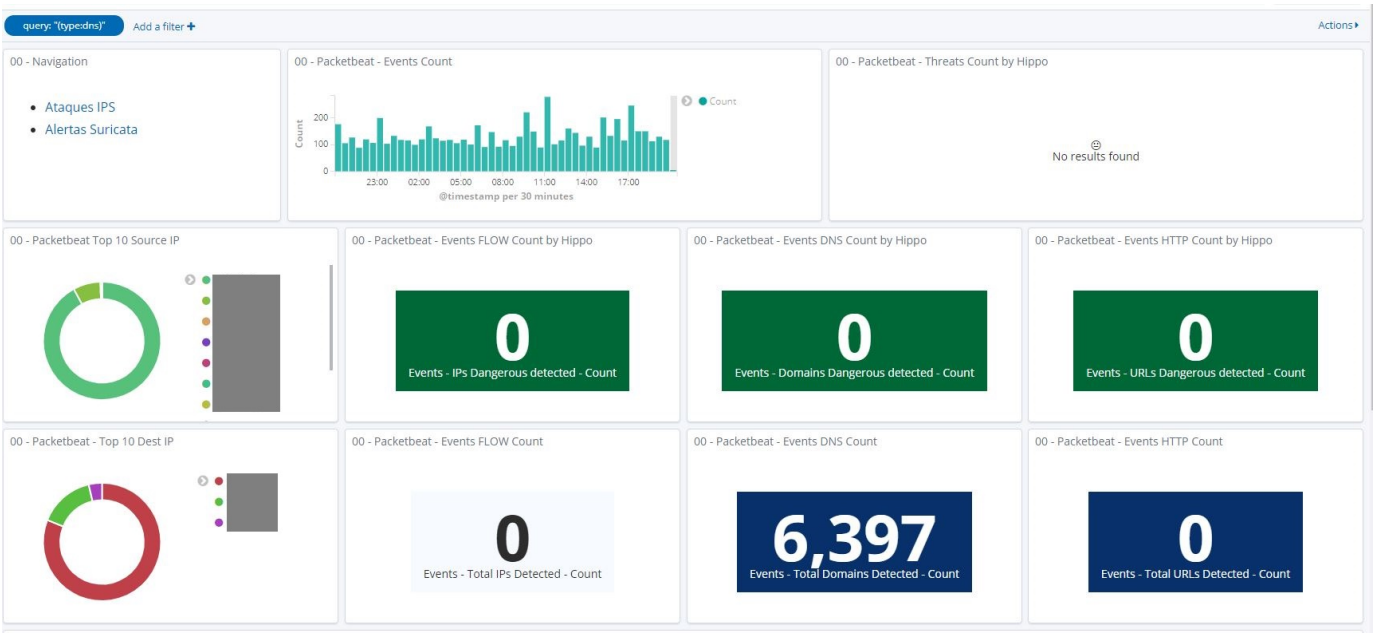


Figura 5.3: Dashboard Packetbeat con Hippocampe filtrando por eventos de tipo dns.

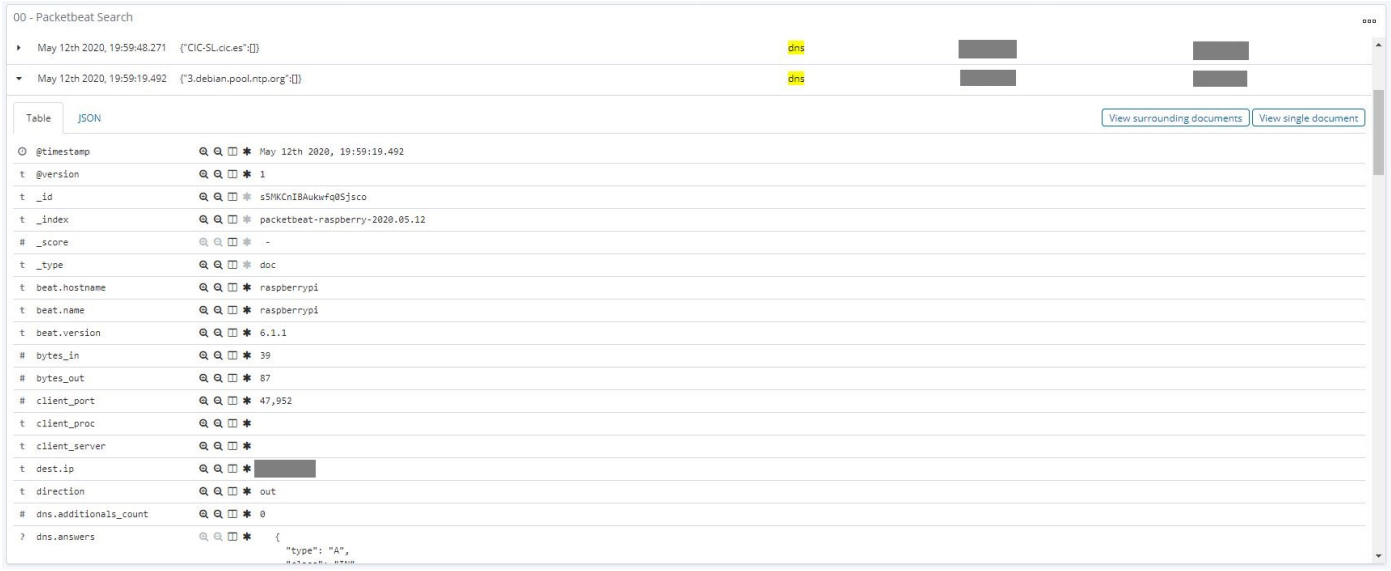


Figura 5.4: Dashboard Packetbeat con Hippocampe con el detalle de un evento de tipo dns 1.

Figura 5.5: Dashboard Packetbeat con Hippocampe con el detalle de un evento de tipo dns 2.

Figura 5.6: Dashboard Packetbeat con Hippocampe con el detalle de un evento de tipo dns 3.

5.2. Dashboard IPS con Hippocampe

En esta sección se muestra el panel elaborado para la fuente de datos del IPS del firewall de CIC. Los eventos generados por el IPS son paquetes o tráfico bloqueado por la tecnología empleada, proporcionando de manera similar a Suricata, la firma o el tipo de ataque detectado en el tráfico. Además, como se puede observar en las visualizaciones presentadas en las figuras 5.7, 5.8 y 5.9, los eventos proporcionan información relativa a la localización geográfica de las direcciones IP que originan el ataque. Al igual que en panel anterior, también se ha creado un histograma, se muestran los eventos (tráfico de red que el IPS ha detectado como peligroso y ha bloqueado) totales enviados por el IPS al ELK (297) y los eventos cuyos observables se han detectado en Hippocampe (12) en las últimas 24 horas. Hippocampe puede aportar mayor información de estas direcciones obtenida de las *feeds* dadas de alta. Por ello es importante integrar fuentes en Hippocampe que almacenen no solo los observables maliciosos, sino también información de éstos, como los tipos de ataque asociados, servicios o protocolos empleados. En la figura 5.10 se expone parte del contenido detallado de un evento desde la visualización de tipo búsqueda (imagen 5.9), donde se observa que Hippocampe ha detectado la dirección IP 192.108.66.211 con un hipposcore de -17.64 en la fuente `http://cinsscore.com/list/ci-badguys.txt`. Esta *feed* no proporciona más información de los observables.

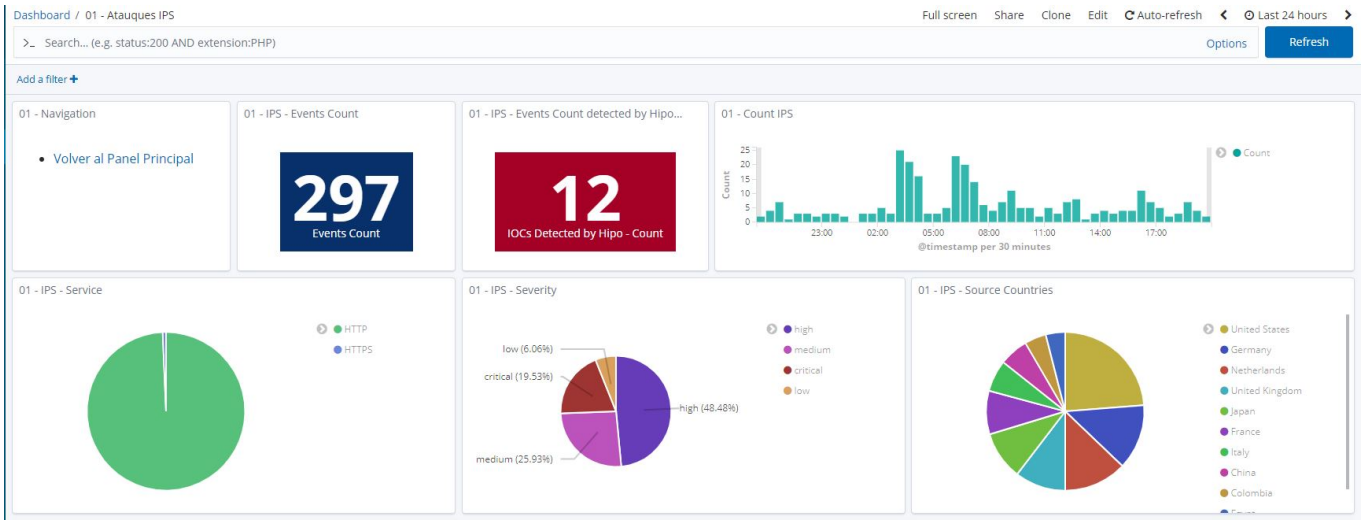


Figura 5.7: Dashboard IPS con los eventos recibidos del IPS y los detectados por Hippocampe. Se muestran diagramas de sectores de los eventos clasificados por servicios, severidad y país origen.

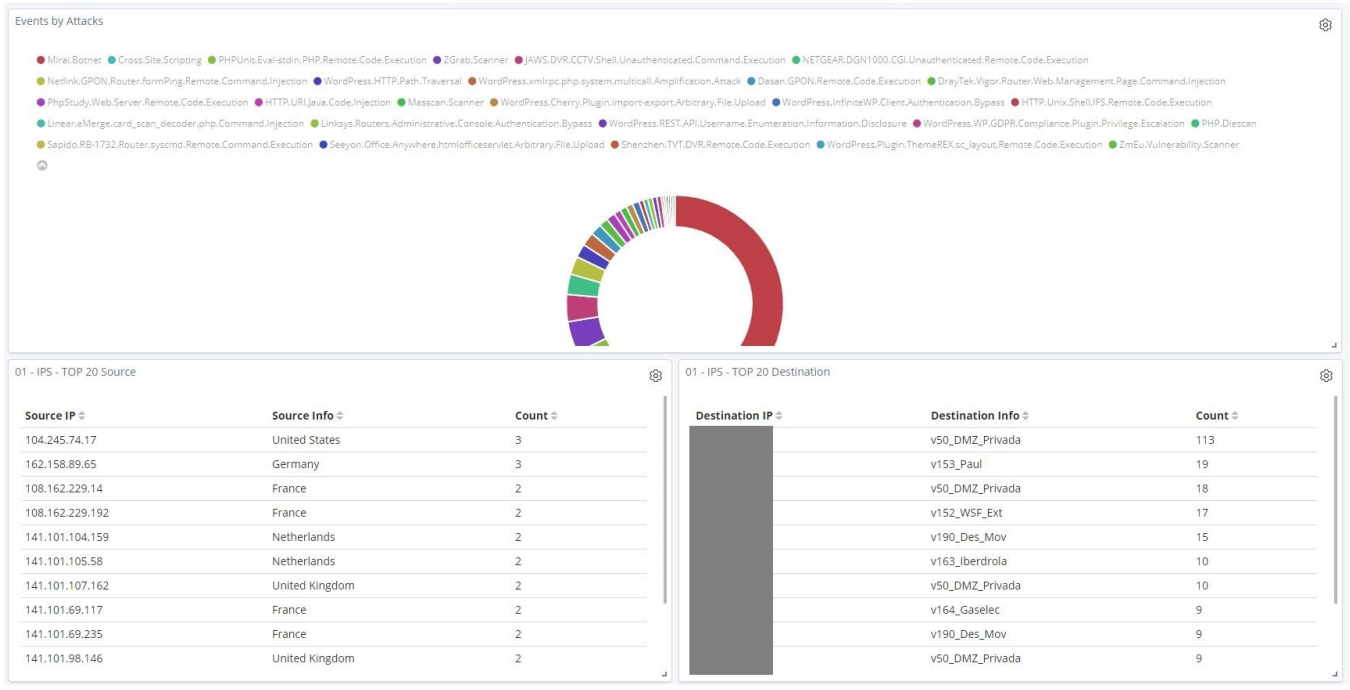


Figura 5.8: Dashboard IPS con los eventos clasificados por el tipo de ataque, direcciones IP origen y destino.

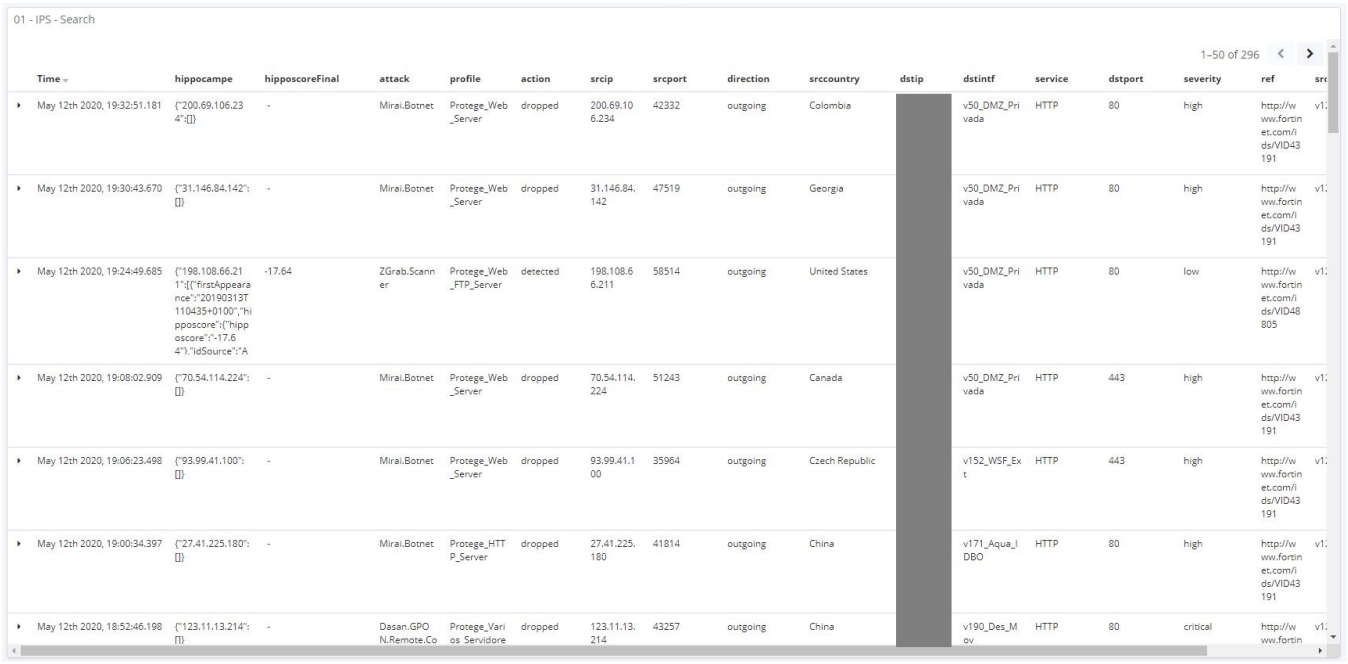


Figura 5.9: Dashboard IPS con el detalle de los eventos.

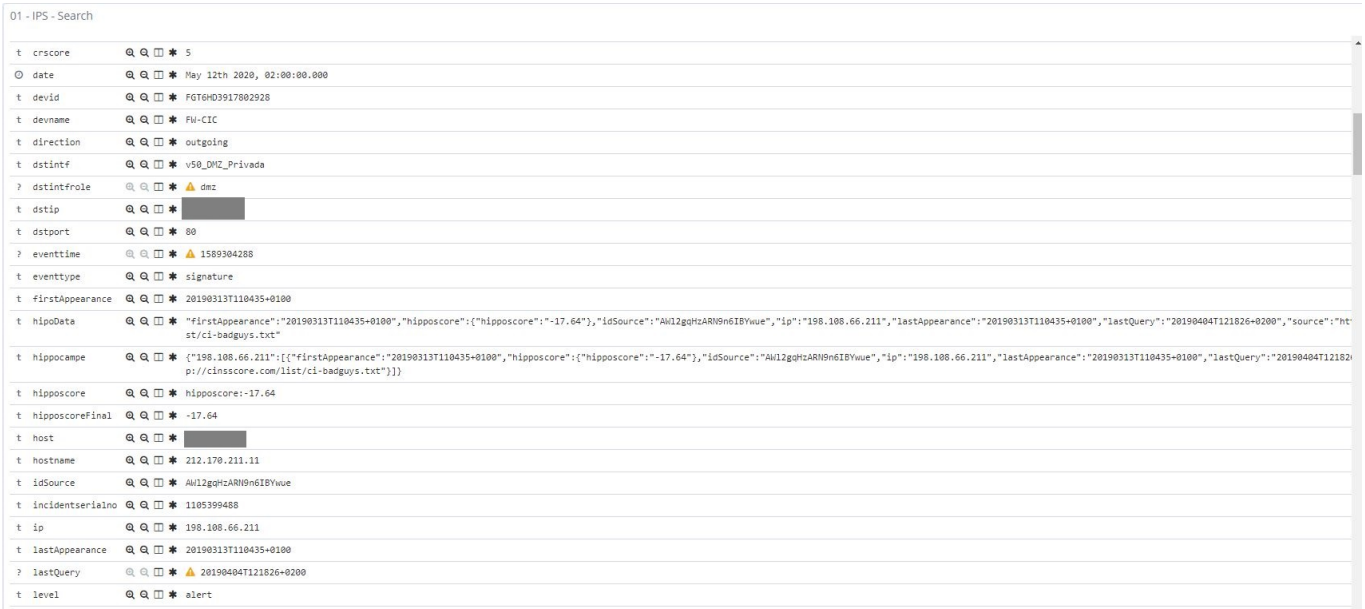


Figura 5.10: Dashboard IPS con el detalle de un evento.

5.3. Dashboard Tráfico de red: Suricata

En esta última sección se expone el panel creado para el tráfico de red analizado por Suricata (figuras 5.11, 5.12 y 5.13). De manera semejante a los *dashboards* anteriores, se han creado dos histogramas y dos métricas donde se aprecian el número total de eventos y el número total de eventos de tipo amenaza a lo largo del tiempo. Además, se han elaborado diagramas de sectores en función de campos como la firma de Suricata detectada, la categoría de la amenaza, el protocolo o la severidad de las alertas. También se ha añadido una visualización de tipo búsqueda para poder analizar con detalle todos los campos de los eventos (en este caso, solo se muestran las alertas).

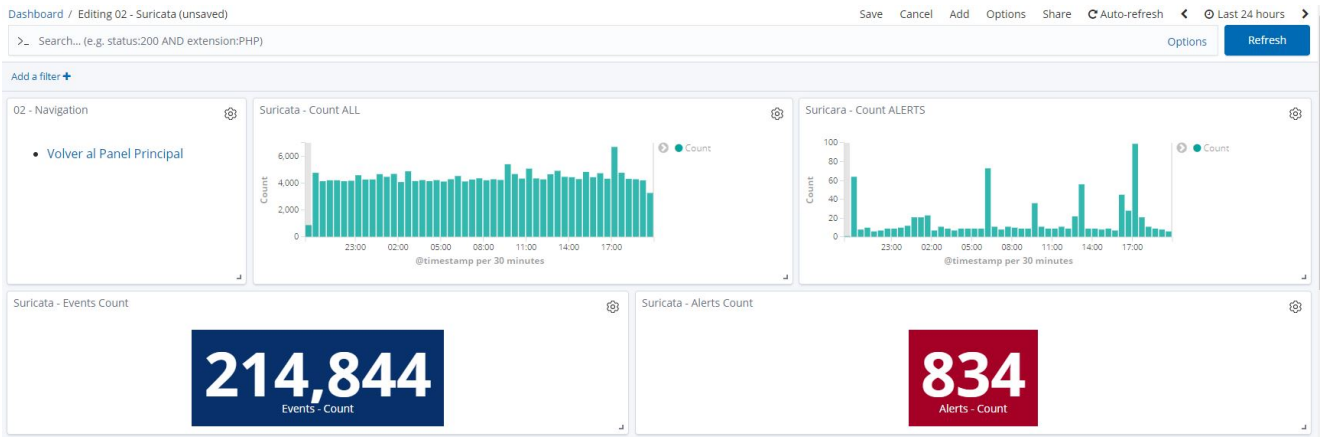


Figura 5.11: Dashboard Suricata con el número de eventos recibidos y alertas detectadas.

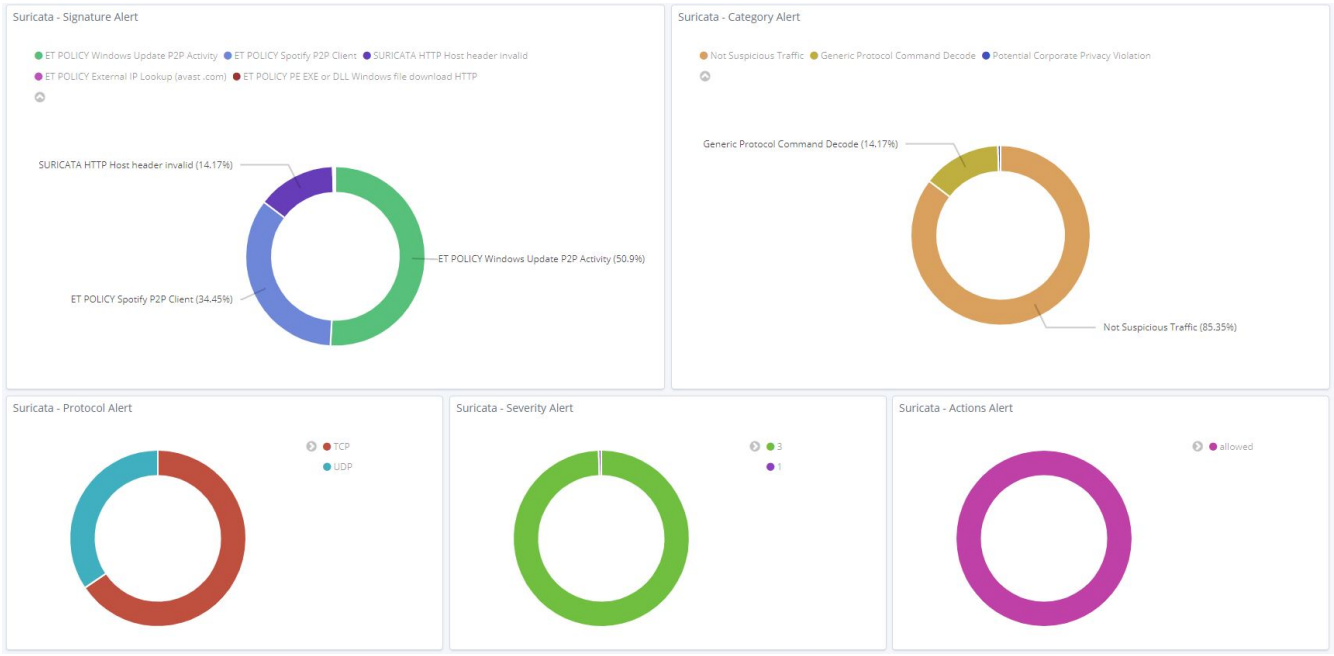
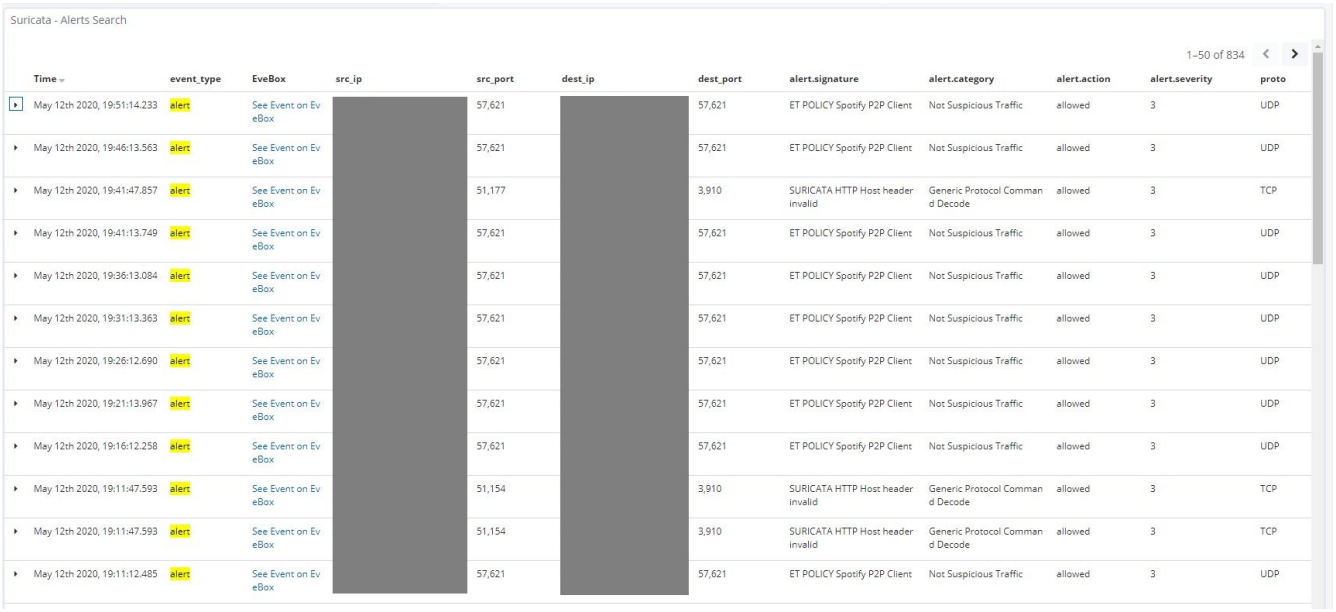


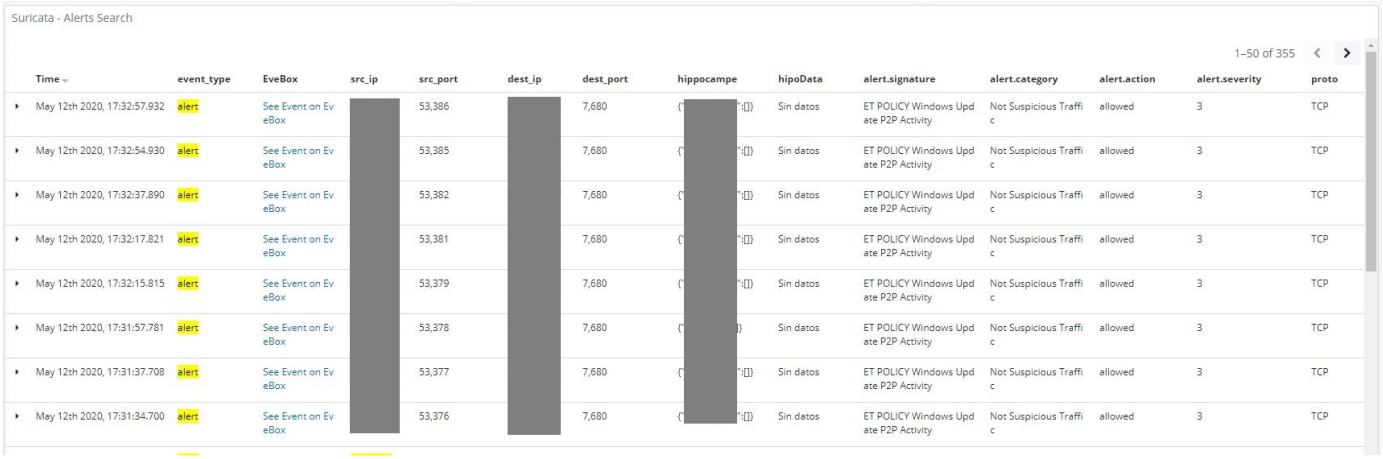
Figura 5.12: Dashboard Suricata con los eventos clasificados por el tipo de alerta, categoría, protocolo, severidad y acción.



Time	event_type	EveBox	src_ip	src_port	dest_ip	dest_port	alert.signature	alert.category	alert.action	alert.severity	proto
May 12th 2020, 19:51:14.233	alert	See Event on EveBox	[redacted]	57,621	[redacted]	57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:46:13.563	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:41:47.857	alert	See Event on EveBox		51,177		3,910	SURICATA HTTP Host header invalid	Generic Protocol Command Decode	allowed	3	TCP
May 12th 2020, 19:41:13.749	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:36:13.084	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:31:13.363	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:26:12.690	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:21:13.967	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:16:12.258	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP
May 12th 2020, 19:11:47.593	alert	See Event on EveBox		51,154		3,910	SURICATA HTTP Host header invalid	Generic Protocol Command Decode	allowed	3	TCP
May 12th 2020, 19:11:47.593	alert	See Event on EveBox		51,154		3,910	SURICATA HTTP Host header invalid	Generic Protocol Command Decode	allowed	3	TCP
May 12th 2020, 19:11:12.485	alert	See Event on EveBox		57,621		57,621	ET POLICY Spotify P2P Client	Not Suspicious Traffic	allowed	3	UDP

Figura 5.13: Dashboard Suricata con el detalle de los eventos.

A modo de ejemplo, se ha aplicado un filtro sencillo en Kibana buscando por una de las direcciones IP interna que ha generado una alerta. De esta manera, se actualizan todos los gráficos del panel mostrando únicamente los eventos con dicha dirección. Como se aprecia en las figuras 5.14 y 5.15, se actualiza todo el panel y se observa que esa IP origen genera la alerta o firma *ET POLICY Windows Update P2P Activity*, categorizado como tráfico no sospechoso con severidad 3. Esta firma detecta la existencia de un PC con las actualizaciones *peer-to-peer* de Windows habilitadas. Esta característica permite descargar una actualización de Windows en un equipo una vez, y emplear esa máquina como servidor para difundir la actualización a todos los PCs de la red local, disminuyendo el ancho de banda empleado de descarga en internet en toda la red. Sin embargo, de manera predeterminada esta configuración no solo actúa en área local, si no que se comparte con otras máquinas a través de internet, por lo que aprovechar estas máquinas fuera de una red local para actualizar Windows 10 puede considerarse un riesgo de seguridad. Además, un posible efecto secundario sería el alto uso del ancho de banda en el PC que actúaría como servidor de las actualizaciones. Por estos motivos, Suricata (empleando las firmas de *Emerging Threats*) alerta sobre esta actividad de tipo *POLICY*, ya que podría existir un incumplimiento de la política de la organización si se prohíbe este tipo de configuraciones en los equipos.



Time	event_type	EveBox	src_ip	src_port	dest_ip	dest_port	hippocampe	hipoData	alert.signature	alert.category	alert.action	alert.severity	proto
May 12th 2020, 17:32:57.932	alert	See Event on EveBox	[redacted]	53,386	[redacted]	7,680	[redacted]	Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP
May 12th 2020, 17:32:54.930	alert	See Event on EveBox		53,385		7,680		Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP
May 12th 2020, 17:32:37.890	alert	See Event on EveBox		53,382		7,680		Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP
May 12th 2020, 17:32:17.821	alert	See Event on EveBox		53,381		7,680		Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP
May 12th 2020, 17:32:15.815	alert	See Event on EveBox		53,379		7,680		Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP
May 12th 2020, 17:31:57.781	alert	See Event on EveBox		53,378		7,680		Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP
May 12th 2020, 17:31:37.708	alert	See Event on EveBox		53,377		7,680		Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP
May 12th 2020, 17:31:34.700	alert	See Event on EveBox		53,376		7,680		Sin datos	ET POLICY Windows Update P2P Activity	Not Suspicious Traffic	allowed	3	TCP

Figura 5.14: Dashboard Suricata con el detalle de los eventos filtrando por una dirección IP.

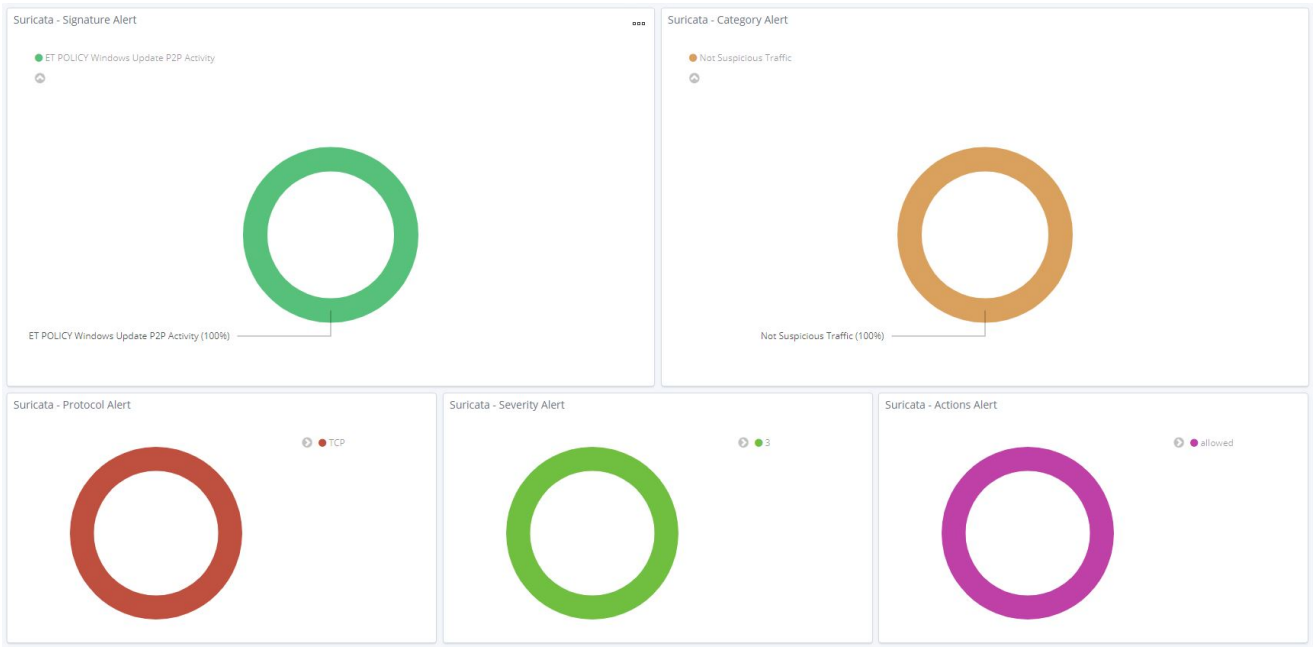


Figura 5.15: Dashboard Suricata con los eventos clasificados por el tipo de alerta, categoría, protocolo, severidad y acción filtrando por una dirección IP.

Capítulo 6

Conclusiones y trabajos futuros

En este último y breve capítulo de la memoria se explican las conclusiones obtenidas de la elaboración del trabajo, así como los conocimientos adquiridos. También se exponen posibles mejoras en la plataforma para aumentar las características y funcionalidades que ofrece la herramienta desarrollada.

6.1. Conclusiones

La finalidad de este proyecto es desplegar una plataforma de ciberseguridad basada en el conjunto ELK para mejorar la inteligencia de amenazas. De esta manera, se pretende identificarlas y aportar un mayor conocimiento útil de las mismas a los operadores y analistas de seguridad informática apoyándoles a concretar las acciones oportunas para hacer frente a los peligros existentes. Además, la herramienta debe garantizar una alta disponibilidad, una recolección masiva de datos y su visualización cercana al tiempo real, y disponer de medidas de autenticación. Gracias a toda la infraestructura desplegada y software instalado y configurado, se puede considerar el cumplimiento de los principales objetivos del trabajo. Por un lado, la herramienta logra consumir una gran cantidad de datos mediante distintos sensores de ciberseguridad, los enriquece con software de información de amenazas y ofrece una búsqueda ágil y visualización en tiempo real con Kibana. Por otro lado, mediante el uso de una infraestructura en clúster en un entorno virtualizado con cuatro máquinas virtuales Linux repartidas en dos nodos, se consigue asegurar una alta disponibilidad del sistema y tolerar el fallo de uno de los dos servidores, y disponiendo además de medidas de autenticación con Keycloak.

La realización del presente proyecto ha proporcionado nuevos y profundos conocimientos en la tecnología Elasticsearch, Logstash y Kibana, muy utilizada en entornos empresariales para el almacenamiento, filtrado y búsqueda de grandes cantidades de datos. Además, se ha aumentado los conocimientos de sistemas Linux y se han desarrollado habilidades para solventar los problemas presentados durante el despliegue, integración y configuración de los componentes que forman la plataforma. También se ha aprendido el lenguaje de scripting Ruby para la integración con servicios mediante API REST, se ha comprendido la importancia del *clustering* y se han adquirido conocimientos del funcionamiento de diversas herramientas de ciberseguridad. Además, se ha conocido el modo de trabajo y diversas herramientas empleadas en un entorno empresarial.

6.2. Trabajos futuros

La plataforma desarrollada cumple con la finalidad y requisitos del proyecto, pero es posible aumentar sus funcionalidades actuales y mejorar ciertos aspectos para conseguir adaptarla a necesidades más específicas.

Por un lado, en lo referente a los servicios de inteligencia de amenazas empleados, se pueden buscar e integrar más fuentes en Hippocampe con mayor información sobre los elementos consultados, y valorar el uso de *feeds* con suscripción. Además, existen otros software complementarios de inteligencia de amenazas que podrían añadirse a la plataforma, como Cortex [47]. Además de Hippocampe,

TheHive Project ofrece esta potente herramienta de análisis de observables, con la principal ventaja de su integración con decenas de analizadores de ciberseguridad (VirusTotal, GoogleDNS, Hippocampe, Fortiguard...) para obtener información de éstos sobre cada observable consultado. También es recomendable comprobar posibles falsos positivos de las alertas de Suricata y desactivarlas, y deshabilitar aquellas alertas consideradas como no peligrosas por y para la organización, notificando así únicamente las realmente relevantes. Además, puede resultar interesante crear reglas personalizadas en función del direccionamiento IP de los sistemas y su criticidad, o alertar sobre rangos de IP específicos.

Por otro lado, se pueden mejorar las medidas de autenticación. Actualmente, se emplea Keycloak como herramienta para securizar el acceso a la interfaz web mediante la creación de usuarios en su base de datos, pero este software ofrece funcionalidades más potentes. Una de las más atractivas es la conexión con LDAP y Directorio Activo, por lo que sería realmente interesante realizar una integración con el Directorio Activo de la organización de manera que sea posible autenticarse con los usuarios corporativos y establecer permisos en función de los grupos y privilegios de cada uno.

Finalmente, se debe tener en cuenta el mantenimiento y escalado de la plataforma. Es posible que en algún momento se decida integrar nuevas fuentes de datos y aumente la carga computacional o de almacenamiento, y esto implique un incremento de los requerimientos hardware de la infraestructura para mantener su correcto y óptimo funcionamiento. En este caso, sería necesario valorar un escalado horizantal (adición de MVs en el clúster), lo cual es una tarea sencilla gracias a la capacidad de escalado ofrecida por ELK, o aumentar las capacidades de procesamiento o almacenamiento de las máquinas virtuales actuales.

Bibliografía

[1] D. Marshall, W. A. Reynolds, and D. McCrory, *Advanced server virtualization: VMware and Microsoft platforms in the virtual data center*. Auerbach Publications, 2006.

[2] B. Hundley, "Optimizing elasticsearch: How many shards per index?." `https://qbox.io/blog/optimizing-elasticsearch-how-many-shards-per-index`. Accessed: 2020-02-03.

[3] F. de Villamil, "Designing the perfect elasticsearch cluster: the (almost) definitive guide." `https://thoughts.t37.net/designing-the-perfect-elasticsearch-cluster-the-almost-definitive-guide-e614eabc1a87`. Accessed: 2020-02-03.

[4] "Stashing your first event." `https://www.elastic.co/guide/en/logstash/current/first-event.html`. Accessed: 2020-08-03.

[5] "Hipposcore." `https://github.com/TheHive-Project/Hippocampe/blob/master/docs/hipposcore.md`. Accessed: 2020-04-21.

[6] D. Schatz, R. Bashroush, and J. Wall, "Towards a more representative definition of cyber security," *Journal of Digital Forensics, Security and Law*, vol. 12, no. 2, p. 8, 2017.

[7] M. Bromiley, "Threat intelligence: What it is, and how to use it effectively," *SANS Institute InfoSec Reading Room*, vol. 15, 2016.

[8] A. Białecki, R. Muir, G. Ingersoll, and L. Imagination, "Apache lucene 4," in *SIGIR 2012 workshop on open source information retrieval*, p. 17, 2012.

[9] "Keycloak relational database setup." `https://www.keycloak.org/docs/latest/server_installation/index.html#_database`. Accessed: 2019-04-18.

[10] A. Leslie, "Nginx vs apache." `https://www.hostingadvice.com/how-to/nginx-vs-apache/`. Accessed: 2019-04-22.

[11] "Elasticsearch: Optimization guide." `https://octoperf.com/blog/2018/09/21/optimizing-elasticsearch/`. Accessed: 2020-03-30.

[12] "Set up elasticsearch." `https://www.elastic.co/guide/en/elasticsearch/reference/6.6/setup.html`. Accessed: 2020-03-27.

[13] "Zen discovery." `https://www.elastic.co/guide/en/elasticsearch/reference/6.6/modules-discovery-zen.html`. Accessed: 2019-11-14.

[14] O. Procopiuc, P. K. Agarwal, L. Arge, and J. S. Vitter, "Bkd-tree: A dynamic scalable kd-tree," in *International Symposium on Spatial and Temporal Databases*, pp. 46–65, Springer, 2003.

[15] C. Dahlqvist, "¿cuántos shards debo tener en mi cluster de elasticsearch?." `https://www.elastic.co/es/blog/how-many-shards-should-i-have-in-my-elasticsearch-cluster`. Accessed: 2020-03-05.

[16] "All specjbb2015 results published by spec." `https://www.spec.org/jbb2015/results/jbb2015.html`. Accessed: 2019-12-17.

[17] "Thread pool." https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-threadpool.html#modules-threadpool. Accessed: 2019-12-18.

[18] "Compressed oops." https://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html. Accessed: 2020-01-08.

[19] "Elasticsearch 6.6.2." https://www.elastic.co/es/downloads/past-releases/elasticsearch-6-6-2. Accessed: 2020-1-13.

[20] "Install elasticsearch with rpm." https://www.elastic.co/guide/en/elasticsearch/reference/6.6/rpm.html. Accessed: 2020-1-13.

[21] "Input plugins." https://www.elastic.co/guide/en/logstash/6.6/input-plugins.html. Accessed: 2020-08-03.

[22] "Filter plugins." https://www.elastic.co/guide/en/logstash/6.6/filter-plugins.html. Accessed: 2020-08-03.

[23] "Output plugins." https://www.elastic.co/guide/en/logstash/6.6/output-plugins.html. Accessed: 2020-08-03.

[24] "Logstash 6.6.0." https://www.elastic.co/es/downloads/past-releases/logstash-6-6-0. Accessed: 2020-08-03.

[25] "Kibana 6.6.0." https://www.elastic.co/es/downloads/past-releases/kibana-6-6-0. Accessed: 2020-08-03.

[26] "Setting up an nginx reverse proxy." https://linuxize.com/post/nginx-reverse-proxy/. Accessed: 2020-03-15.

[27] "Keycloak downloads - 4.8.3.final." https://www.keycloak.org/archive/downloads-4.8.3.html. Accessed: 2020-03-22.

[28] "Wildfly admin guide." http://docs.wildfly.org/18/Admin_Guide.html. Accessed: 2020-03-22.

[29] "Wildfly 10 - authentication failure in domain mode." https://stackoverflow.com/questions/43620988/wildfly-10-authentication-failure-in-domain-mode. Accessed: 2020-03-27.

[30] "Selinux configuration." https://galeracluster.com/library/documentation/selinux.html. Accessed: 2020-03-28.

[31] "Everything you need to install mariadb for development and production." https://mariadb.com/downloads/#connectors. Accessed: 2019-04-18.

[32] "keycloak-mariadb." https://github.com/mposolda/keycloak-mariadb. Accessed: 2020-03-30.

[33] "Move keycloak database from h2 to mysql." https://github.com/CodepediaOrg/bookmarks.dev/wiki/Move-keycloak-database-from-H2-to-MySql. Accessed: 2020-03-30.

[34] "Relational database setup." https://www.keycloak.org/docs/latest/server_installation/#_database. Accessed: 2020-03-30.

[35] "Configuration of keycloak server in performance testsuite fails." https://issues.redhat.com/browse/KEYCLOAK-8632. Accessed: 2020-03-30.

[36] "Control de acceso http (cors)." https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS. Accessed: 2020-04-04.

[37] "A keycloak authorization plugin for kibana." https://github.com/novomatic-tech/keycloak-kibana. Accessed: 2020-04-03.

[38] "Identifying client ip addresses." https://www.keycloak.org/docs/7.0/server_installation/#identifying-client-ip-addresses. Accessed: 2020-04-04.

[39] R. Fratkina, "Kibana plugin api changes in 6.6." https://www.elastic.co/es/blog/kibana-plugin-api-changes-in-6-6. Accessed: 2020-04-04.

[40] "Virtual router redundancy protocol (vrrp) version 3 for ipv4 and ipv6." https://tools.ietf.org/html/rfc5798. Accessed: 2020-04-10.

[41] "Install filebeat on raspberry pi 3." http://www.programmersought.com/article/31601555670/. Accessed: 2020-04-12.

[42] "Beats - lightweight shippers for elasticsearch y logstash." https://github.com/elastic/beats. Accessed: 2020-04-12.

[43] "Rules emerging threats." https://rules.emergingthreats.net/open/suricata-4.0/rules/. Accessed: 2020-04-18.

[44] "Install guide." https://github.com/TheHive-Project/Hippocampe/blob/master/docs/install_guide.md. Accessed: 2020-04-25.

[45] "Feeds hippocampe." https://github.com/TheHive-Project/Hippocampe/tree/master/core/conf/feeds. Accessed: 2020-04-27.

[46] "Elastalert - easy and flexible alerting with elasticsearch." https://github.com/Yelp/elastalert. Accessed: 2020-05-09.

[47] "Cortex: a powerful observable analysis and active response engine." https://github.com/TheHive-Project/Cortex. Accessed: 2020-05-31.

Anexos

Anexo A

Configuración Nginx

Fichero `/etc/nginx/conf.d/kibana.conf`:

```
server {
        listen       10.10.10.3:80;
        server_name  _;
        location / {
            proxy_pass http://127.0.0.1:5601;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }
    }
```

Anexo B

Configuración publicación Keycloak

Fichero `/opt/keycloak/domain/configuration/host-master.xml`:

```
...
<interfaces>
        <interface name="management">
            <inet-address value="${jboss.bind.address.management:0.0.0.0}"/>
        </interface>
        <interface name="public">
            <inet-address value="${jboss.bind.address:0.0.0.0}"/>
        </interface>
</interfaces>
...
```

Anexo C

Configuración Proxy Keycloak

Fichero `/opt/keycloak-proxy/config/proxy.json`:

```
{
    "target-url": "http://127.0.0.1:5601",
    "send-access-token": false,
    "bind-address": "127.0.0.1",
    "http-port": "8081",
    "https-port": "8082",
    "applications": [
        {
            "base-path": "/",
            "adapter-config": {
                "realm": "Kibana",
                "auth-server-url":"http://10.60.3.67:8080/auth",
                "realm-public-key": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A/
                   MIIBCgKCAQEAjBTvKElZ5rDZZDFVSK38+sr6w4bp/bz1onSPrQ/
                   paDBovaTU1x5xqnuheiZHvQIDAQAB",
                "resource": "kibana",
                "ssl-required": "none",
                "credentials": {
                        "secret": "cc9caa28-5514-4ce8-9874-45
                           be941b88b2"
                }

            },
            "constraints": [
                {
                    "pattern": "/*",
                    "authenticate": true,
                                "permit-and-inject": false
                }
            ],
                    "proxy-address-forwarding": true
         }
    ],
        "header-names": {
                "keycloak-username": "X-Forwarded-User",
                "keycloak-role": "X-Forwarded-Role"
        }
}
```

Anexo D

Cambios fichero 'routes.js' para logout

Fichero `/usr/share/kibana/plugins/logout/server/routes.js` para la versión 6.4.0:

```
export default function (server) {
   server.route({
        path: '/api/logout/logout',
        method: 'GET',
        handler(req, reply) {
                const config = req.server.config();
                var respuesta = new Object();
                var name = config.get('logout.headerName');
                respuesta.server_Url = config.get('logout.keycloakUrl');
                respuesta.proxy_Url = config.get('logout.proxyUrl');
                respuesta.realm = config.get('logout.realm');
                respuesta.client = config.get('logout.client');
                respuesta.username = req.headers[ name ];
                reply(respuesta);
         }
    });
};
```

Fichero `/usr/share/kibana/plugins/logout/server/routes.js` modificado para la versión 6.6.0:

```
export default function (server) {
   server.route({
        path: '/api/logout/logout',
        method: 'GET',
        async handler(req, h) {
                const config = req.server.config();
                var respuesta = new Object();
                var name = config.get('logout.headerName');
                respuesta.server_Url = config.get('logout.keycloakUrl');
                respuesta.proxy_Url = config.get('logout.proxyUrl');
                respuesta.realm = config.get('logout.realm');
                respuesta.client = config.get('logout.client');
                respuesta.username = req.headers[ name ];
                return await respuesta;
         }
    });
};
```

Anexo E

Configuración HAProxy

Fichero `/etc/haproxy/haproxy.cfg`:

```
global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon
    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats
#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#---------------------------------------------------------------------
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000
#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------
frontend  http-incoming
        mode http
        bind *:80
        stats uri /haproxy?stats
        default_backend nginx_pool
        option forwardfor
        http-request set-header X-Forwarded-Port 8085
        http-request set-header X-Forwarded-For %[src]
        http-request set-header X-Forwarded-Proto http

```

```
45 #---------------------------------------------------------------------
46 # static backend for serving up images, stylesheets and such
47 #---------------------------------------------------------------------
48 backend nginx_pool
49     balance     source
50     mode http
51     server  websvr1 10.60.3.67:8085 weight 1 maxconn 3000 check
52     server  websvr2 10.60.3.68:8085 weight 1 maxconn 3000 check
```

Anexo F

Configuración Keppalived

Fichero `/etc/haproxy/haproxy.cfg` (máster):

```
global_defs {
   #notification_email {
     #sysadmin@firewall.loc
   #}
   #notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server localhost
    smtp_connect_timeout 30
}

vrrp_script chk_haproxy {
  script "pidof haproxy"  # check the haproxy process
  interval 2 # every 2 seconds
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 200
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    unicast_src_ip 10.10.10.3
    unicast_peer {
      10.10.10.4
   }
    virtual_ipaddress {
        10.10.10.5/20
    }
    track_script {
        chk_haproxy
  }
}
```

Anexo G

Configuración Logstash: Ejemplo input y output

```
input {
  beats {
    port => 5043
  }
  beats {
    port => 5046
    add_field => { "origen-evento" => "suricata" }
  }
  udp {
    port => 5045
    type => "syslog"
  }
}

output {
  if ([origen-evento]=="suricata") {
    elasticsearch {
      hosts => ["10.10.10.2:9200"]
      index => "suricata-%{+YYY.MM.dd}"
    }
  }
  else if ([type]=="syslog") {
    elasticsearch {
      hosts => ["10.10.10.2:9200"]
      index => "syslog-%{+YYY.MM.dd}"
    }
  }
  else {
    elasticsearch {
      hosts => ["10.10.10.2:9200"]
      index => "packetbeat-%{+YYY.MM.dd}"
    }
  }
}
```

Anexo H

Fuentes habilitadas en Hippocampe

Feed	Type	URL
GreenSnow	ip	http://blocklist.greensnow.co/greensnow.txt
CINS Active Threat Intelligence	ip	http://cinsscore.com/list/ci-badguys.txt
IP brute force blocker list	ip	http://danger.rulez.sk/projects/bruteforceblocker/blist.php
Sourcefire VRT	ip	http://labs.snort.org/feeds/ip-filter.blf
Malc0de	ip	http://malc0de.com/bl/IP_Blacklist.txt
Malwaredomains	domain	http://mirror1.malwaredomains.com/files/domains.txt
Alienvault IP reputation list	ip	http://reputation.alienvault.com/reputation.data
Emerging Threats	ip	http://rules.emergingthreats.net/blockrules/compromised-ips.txt
Binarydefense	ip	http://www.binarydefense.com/banlist.txt
DShield High Level	domain	http://www.dshield.org/feeds/suspiciousdomains_High.txt
DShield Medium Level	domain	https://dshield.org/feeds/suspiciousdomains_Medium.txt
DShield Low Level	domain	https://dshield.org/feeds/suspiciousdomains_Low.txt
Malwaredomainlist	domain	http://www.malwaredomainlist.com/hostslist/hosts.txt
Tor Project	ip	https://check.torproject.org/torbulkexitlist
Abuse Feodo Tracker	ip	https://feodotracker.abuse.ch/downloads/ipblocklist.txt
OpenPhish	url	https://openphish.com/feed.txt
StevenBlack/hosts	domain	https://raw.githubusercontent.com/StevenBlack/hosts/master/alternates/fakenews-gambling-porn-social/hosts
Abuse SSL	ip	https://sslbl.abuse.ch/blacklist/sslipblacklist.txt
Badips	ip	https://www.badips.com/get/list/any/2

Anexo I

Script Requests Hippocampe

```
require 'tempfile'
require 'fileutils'
require 'net/http'
require 'uri'
require 'json'

# the value of `params` is the value of the hash passed to `script_params`
# in the logstash configuration
def register(params)
end
# filter runs for every event
# return the list of events to be passed forward
# returning empty list is equivalent to event.cancel
def filter(event)

        event.set("hipoData","")

        # Para el tipo FLOW -> paquetes con reglas -- ip origen -> ip destino
        if  (event.get("type") == "flow")

                # Verificamos si existe una ip origen y destino, ya que si no causa error en hippocampo
                if(!event.get("[dest][ip]"))
                        event.set("hipoData","Sin datos")
                        return []
                end

                uri = URI.parse("http://10.10.10.3:5000/hippocampe/api/v1.0/more")
                request = Net::HTTP::Post.new(uri.path)
                request.content_type = "application/json"
                request.body = JSON.dump({
                        event.get("[dest][ip]") => {
                                        "type" => "ip"
                                          }
                })

                req_options = {
                          use_ssl: uri.scheme == "https",
                }

                response = Net::HTTP.start(uri.hostname, uri.port, req_options) do |http|
                        http.request(request)
```

```
42                  end
43
44                  event.set("hippocampe",response.body)
45
46                  #######
47                  # Quitamos la cabecera que devuelve hippocampe
48                  cabecera=String(response.body)
49                  ip=String(event.get("[dest][ip]"))
50                  tamip=ip.length
51                  tam=tamip+6 # porque hay 5 caracteres mas que sobran
52                  cabecera=cabecera[tam...cabecera.length-4]
53                  #######
54                  event.set("hipoData",cabecera)
55
56          elsif (event.get("type") == "dns")
57                  # Elimino el ultimo caracter porque aparece un punto
58                  miDominio=event.get("[dns][question][name]")
59                  miDominio=miDominio[0 0...miDominio.length-1]
60
61                  uri = URI.parse("http://10.10.10.3:5000/hippocampe/api/v1.0/more
                        ")
62                  request = Net::HTTP::Post.new(uri.path)
63                  request.content_type = "application/json"
64                  request.body = JSON.dump({
65                          miDominio => {
66                                          "type" => "domain"
67                                            }
68                  })
69
70                  req_options = {
71                            use_ssl: uri.scheme == "https",
72                  }
73
74                  response = Net::HTTP.start(uri.hostname, uri.port, req_options)
                        do |http|
75                          http.request(request)
76                  end
77
78                  event.set("hippocampe",response.body)
79
80                  #######
81                  # Quitamos la cabecera que devuelve hippocampe
82                  cabecera=String(response.body)
83                  tamdominio=miDominio.length
84                  tam=tamdominio+6 #porque hay 6 caracteres mas que sobran
85                  cabecera=cabecera[tam...cabecera.length-4]
86                  #######
87                  event.set("hipoData",cabecera)
88
89          elsif (event.get("type")=="http")
90
91                  # Necesitamos la URL completa
92                  url=event.get("[http][request][headers][host]")
93                  path=event.get("path")
94                  url="http://#{url}#{path}"
95
96                  uri = URI.parse("http://10.10.10.3:5000/hippocampe/api/v1.0/more
                        ")
97                  request = Net::HTTP::Post.new(uri.path)
98                  request.content_type = "application/json"
99                  request.body = JSON.dump({
```

```
100                         url => {
101                                         "type" => "url"
102                                           }
103                 })
104
105                 req_options = {
106                           use_ssl: uri.scheme == "https",
107                 }
108
109                 response = Net::HTTP.start(uri.hostname, uri.port, req_options)
                       do |http|
110                           http.request(request)
111                 end
112
113                 event.set("hippocampe",response.body)
114
115                 #######
116                 # Quitamos la cabecera que devuelve hippocampe
117                 cabecera=String(response.body)
118
119                 tamurl=url.length
120                 tam=tamurl+6 #porque hay 5 caracteres mas que sobran
121                 cabecera=cabecera[tam...cabecera.length-4]
122                 #######
123                 event.set("hipoData",cabecera)
124
125         end #end if
126
127         #Cada tilde se traduce como 4 carcateres en la respuesta de hippocampo y
                da problemas con las operaciones
128         #de substrings para quitar la cabecera de la respeusta de hippocampo. Se
                propone que si es menor de 20 el tamanho
129         #de la respuesta sin cabecera son las tildes, se pone 20 por si acaso
               hay muchas.
130         if ((event.get("hipoData")=="") or (event.get("hipoData").length<20))
131                 event.set("hipoData", "Sin datos")
132         end
133
134         return [event]
135 end
```

Anexo J

Configuración Logstash: Filter Packetbeat

```
filter {

  if [type]!="suricata" and [type]!="syslog" {

  ruby {
    path => "/home/root/hippoRequest.rb"
  }

  mutate {
    rename => ["source", "sourcetemporal"]
  }

  kv {
    #allow_duplicate_values => false
    source => "hipoData"
    field_split_pattern => ","
    value_split => ":"
    remove_char_key => "\"{}"
    remove_char_value => "\"{}"
  }

  ruby {
    code => '
      score=event.get("hipposcore")
      if score!=nil
        if score.kind_of?(Array) # se verifica si es un array para quitar los
            dos ultimos caracteres
          score=String(score)
          score=score[score.length-8...score.length-2]
          # Verificar si es positivo o negativo para coger un caracter mas para
              el signo negativo
          if (score[0...1]!="-") # es positivo
            score=score[1...score.length] # quitamos el primer caracter porque
                es positivo y nos cogeria los dos puntos
          end

        else # No es un array, por lo que no hay que quitar los 2 ultimos
            caracteres
          score=String(score)
          score=score[score.length-6...score.length]
          # Verificar si es positivo o negativo para coger un caracter mas para
              el signo negativo
```

```
38            if (score[0...1]!="-") # es positivo
39              score=score[1...score.length] # quitamos el primer caracter porque
                    es positivo y nos cogeria los dos puntos
40            end
41          end # end si es o no un array
42          event.set("hipposcoreFinal",score)
43
44        end #end if
45
46        #Modificamos el campo domain para que solo aparezca una vez en caso de
              estar en varias fuentes
47        if (event.get("type")=="dns")
48
49          # Elimino el ultimo caracter porque aparece un punto
50          miDominio=event.get("[dns][question][name]")
51          miDominio=miDominio[00...miDominio.length-1]
52          event.set("domain",miDominio)
53        end
54      '
55    }
56      mutate {
57        rename => ["source", "urlSource"]
58      }
59      mutate {
60        rename => ["sourcetemporal", "source"]
61      }
62    }
63  }
```